A Technical Report
on

Modeling and Implementation of a Packet Routing Switch for Satellite Communications

August 16, 1993

Daniel S. Kaye

Department of

Computer Science and Engineering

Auburn University

Auburn, AL 36849

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		THESIS/~~DISSERTATION~~

4. TITLE AND SUBTITLE
Modeling and Implementation of a Packet Routing Switch for Satellite Communications

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel S. Kaye

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending:
Auburn Alabama (Univ.)

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA-
94-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI
2950 P STREET
WRIGHT-PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distribution Unlimited
MICHAEL M. BRICKER, SMSgt, USAF
Chief Administration

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DTIC QUALITY INSPECTED 5

14. SUBJECT TERMS	15. NUMBER OF PAGES 45
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

A Technical Report
on

Modeling and Implementation of a Packet Routing Switch for Satellite Communications

August 16, 1993

Daniel S. Kaye

Department of

Computer Science and Engineering

Auburn University

Auburn, AL 36849

ABSTRACT

The principal component needed to model a satellite communications network node is a high-speed packet routing switch. The design and testing of routing switches utilizing Transputer networks has been examined extensively in the literature. One such implementation by Khan and Ward made use of cooperating processes scheduled by the Transtech Genesys operating system. The performance of this system was unacceptable with problems attributed to packet duplication and process contention. In this report, we examine an alternate approach using cooperating occam processes without the overhead associated with an operating system.

The routing switch permits dynamic updates to its routing tables from a host computer. Library functions are provided to host processes to offer this service. However, accommodating these updates influences the design of the underlying Transputer network. Allowances must be made not only for expedient routing of messages within the Transputer network but also for rapid and efficient routing table updates within the Transputer network. Results discussed in this report reflect the design and performance of the routing switch. Throughput efficiency under a variety of test conditions is also provided.

Table of Contents

LIST OF FIGURES

1 INTRODUCTION

Space-based communication network resources are too expensive for use in experimental studies, and ground-based testbeds are generally used to provide a cost effective means of studying them. One such testbed was constructed to study laser cross-link targeting and packet routing in low altitude multiple satellite (LAMS) networks (Chow, Newman-Wolfe, Ward and McLochlin 1988). The testbed configuration consists of five fully connected nodes with each node modelling a satellite in the LAMS network. Each satellite (node) will possess four bidirectional laser transceivers with which to establish cross-link communications, as illustrated in Figure 1. In the testbed, each node is modelled using a PC with additional communications hardware. The nodes communicate with one another via bench-mounted retargetable laser transceivers.

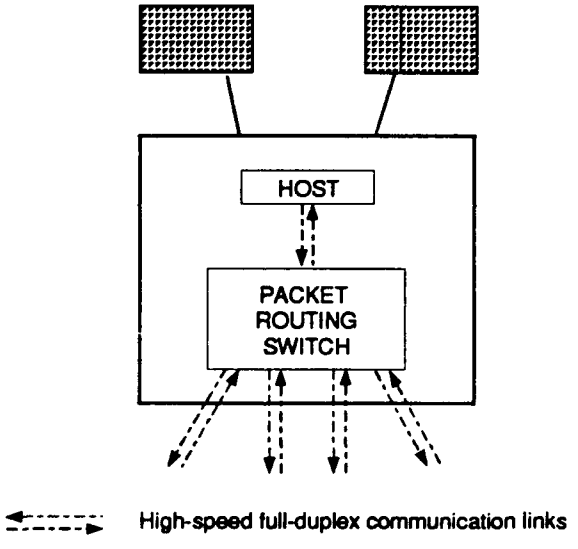


Figure 1 Satellite communication channels.

Communication links between the nodes are required to support full-duplex data rates of at least 1 Mbps to adequately model a LAMS cross-link. Since a Host PC alone could not satisfy the high data rate requirements, a high speed peripheral device with a PC interface was required. The INMOS T800 Transputer system was evaluated in the original design for its ability to achieve the required rates. Transtech's Genesys operating system (Transtech Devices Limited 1990) and high-level language

compilers, coupled with the Transputer's high external data rates were selected as the ideal combination for providing the Host PC's high-speed data communications needs. Details of the implementation are provided in (Khan and Ward 1991; Ward and Khan 1991).

Low throughput rates using the above system rendered it unsuitable for the project testbed. (Ward, Khan, Chow and Newman-Wolfe 1992) Link performance data (Crowell 1990) suggests that the Inmos physical and link layer protocols were not responsible for the poor performance. An alternative explanation for the poor performance is excessive packet duplication required to communicate across intermediate processes and process contention while serving multiple communication channels. With this in mind, this paper evaluates an alternate approach that includes communicating sequentual processes (CSP) (Howe 1985) specifications and an occam implementation that avoids the overhead associated with an operating system.

Section 2 reviews preliminary design considerations relevant to new satellite model development. Section 3 describes the Transputer configuration within the context of the satellite model implementation. Section 4 highlights the functional design of each of the occam processes along with enhancements made during the development process. Throughput statistics for the preliminary and enhanced versions of the new routing switch under a variety of test conditions are presented in Section 5. Comparisons between the original GENESYS, preliminary occam, and enhanced occam implementations are discussed in Section 6.

2 PRELIMINARY DESIGN CONSIDERATIONS

Satellite Network Fundamentals

Satellite networks typically have the following features: one or more satellites placed in geosynchronous orbits; satellites used as communications relays; point-to-point uplinks, with a common channel shared by several earth stations using a combination of FDMA and TDMA; broadcast downlinks; and long propagation delays (in the order of 270ms). Low Altitude Multiple Satellite (LAMS) network systems employ point-to-point laser communications, which satisfy precise targeting, low power, light weight

and small size (so-called SWAP - size, weight and power) constraints. (Paul and Marshalek 1989) Such systems have wide potential in defense and space applications and are characterized as follows:

1. Multiple satellites in a low altitude orbit functioning as store-and-forward Data Communications Equipment (DCE).
2. Point-to-point laser communication with very high bandwidth (20 Mbps - 1 Gbps).
3. Long distance communication links (1,000 km - 10,000 km with relatively high bit error rate (BER, typically $10^{-3} - 10^{-7}$).
4. Limited communication links per satellite due to size, weight and power (SWAP).
5. High mobility.

The laser link channel is characterized by random errors resulting from optical noise sources such as quantum noise, preamplifier thermal noise, dark current noise, detect excess noise, and optical background noise; and by burst errors from beam mispointing and subsequent tracking loss (Paul and Marshalek 1989). The number of satellites in this environment is expected to range from twenty to over a hundred. Each satellite will be in direct (i.e. point-to-point) communication with a few others. In the model currently under investigation this number is constrained to four (i.e. each satellite will have four laser transceivers). A typical network is illustrated in Figure 2. Additional details of LAMS networks may be found in (Ward and Khan 1991, Ward and Choi 1991).

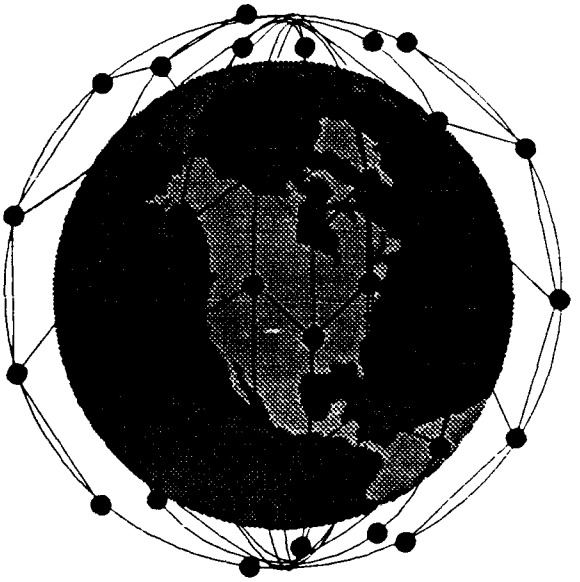


Figure 2 Low altitude multiple satellite (LAMS) network.

Testbed Requirements

One of the many components of a low altitude multiple satellite (LAMS) network is a retargetable communications laser. Initial studies into laser communications, target acquisition, and routing that models a five node LAMS network configuration can be performed using a ground-based testbed. Of these five nodes, only three will actually use laser trackers to communicate with each other; the others will be directly connected through link simulators. These simulators permit a variety of link conditions to be investigated, including bust errors, random errors, and link failure. The testbed model is illustrated in Figure 3.

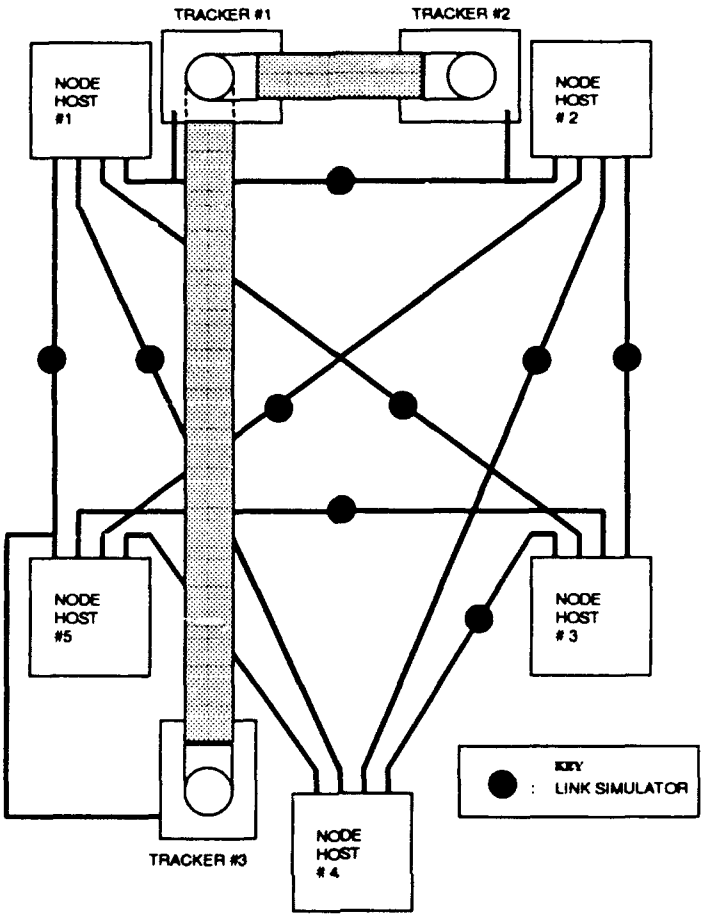


Figure 3 Testbed for five node simulation.

The directly connected links of the satellite nodes require full-duplex communications with data rates of at least 1 Mbps. Since a PC host computer cannot satisfy the high data rate requirements, a high speed peripheral device with a PC interface, such as a Transputer, is needed.

Transputer Architecture Fundamentals

A Transputer is a complete computer on a chip. The Transputer used in this project (an INMOS T800) is composed of a 32 bit processor capable of 10 million instructions per second, a hardware floating point unit, 4 Kbytes of very fast static RAM, a programmable memory interface (which allows up to 4 Gbytes physical memory external to the Transputer) and four bidirectional communication links capable of rates up to 20 Mbps. Each communication link is implemented as an autonomous DMA (Direct Memory

Access) engine so that it can perform communications with external devices as background tasks to the processor with negligible performance degradation. This architecture is reflected in Figure 4.

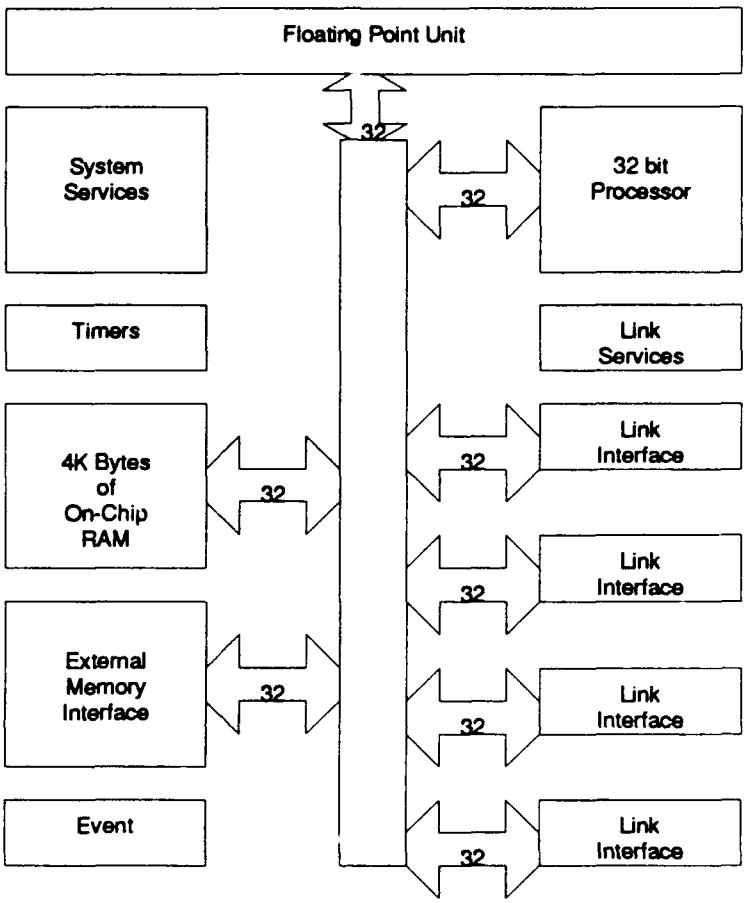


Figure 4 Block diagram of the 32-bit Transputer.

The topology of T800 configurations is constrained by the fixed communication links, allowing flexibility only in selecting available soft link connections. The on-site hardware is an MCP1000 system with four sites (or rows) by four slots (or columns) filled, for a total of sixteen Transputers. Fixed link connections are installed between consecutive Transputers in the same site to connect them together in series. The two remaining links on each transputer are connected to a software-controlled crossbar switch. These "link switches" enable the user to connect any link to any other link that is connected to any switch on the board.

The occam programming language was designed to simplify the task of concurrent programming on netwroks of INMOS transputers. An occam program is made up of a number of processes which can be declared to run sequentially or concurrently. Concurrent processes, which cannot use shared resources, communicate across occam channels. These channels are single direction, point to point connections between processes that provide synchronized message communication.

Transputers are equiped with hardware to support concurrency and message passing via occam channels. A collection of concurrently executing occam processes can be directly mapped onto either one transputer, which shares its time between them, or onto multiple transputers, each taking a subset of the processes.

Synchronized communications between INMOS Transputers use the TRAM link protocol shown in Figure 5. The data format is as follows: Each byte is transmitted as a start bit, followed by a one bit, eight data bits and a stop bit.

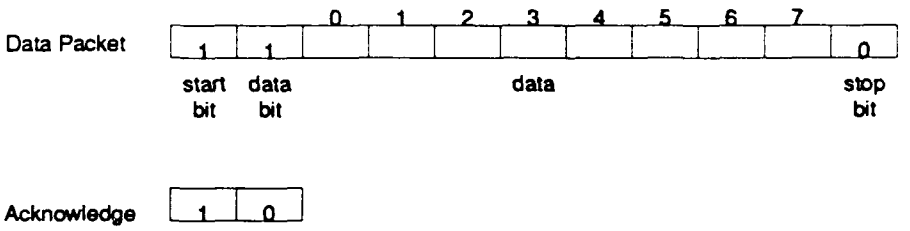


Figure 5 Link protocol

After transmitting a data byte, the sender waits until an acknowledge is received, consisting of a start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged byte, and that the receiving link is able to receive another byte. Acknowledges may not be sent in advance. The receiving end starts with an empty buffer, ready to receive the first byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

A network of INMOS T800 Transputers can satisfy the host computer's high speed data communications needs because it has an inherent capability for high speed, full duplex, synchronous communications, and also because high level language compilers (such as occam) are available for Transputer programming.

Original Simulation Results

The Genesys and C implementation found in (Khan and Ward 1991; Ward and Khan 1991) provides the background for the new work. The general hardware and software architectures were maintained in the new design to permit meaningful comparisons with the results of the original implementation. These architectures are presented in Section 3. Genesys and C implementation test results are provided for comparison with the results presented in Section 5. The following sections describe each of the tests performed, their respective configurations and the recorded results. An attempt is also made to justify the selection of these tests.

Test 0: Maximum Link Throughput The Transputer links are reported to function at 10 or 20 Mbps, as selected by the user. Their throughput was recalculated using the Genesys supplied datalink layer functions so that the throughput results obtained from the routing switch tests could be consistently compared to the maximum data rate. The maximum data rate for the Transputer links was computed by sending messages from one TRAM to an adjacent TRAM over a specified link. The source transmitted 5000 1002-byte packets at constant packet delay intervals. The maximum throughput was calculated by decreasing the packet transmitting delay on successive runs of the test. The received messages were timed and the throughput was calculated. The maximum throughput recorded from the test was $\lambda_{max} = 0.707\ Mbps$.

Although the maximum data rate of the Transputer links is easy to calculate, the accumulation of meaningful performance results for the Transputer based router is more complicated. This difficulty stems from the fact that there are numerous modes of operation in which these results could be accumulated. Because some of these modes of operation yield redundant information, tests were performed on strategically identified configurations. These cases were chosen because they would yield valuable information in determining the routing switch's performance in its original testbed environment.

Test 1: Maximum Switch Throughput The first test involves recording the throughput of the packet routing switch under ideal conditions. The purpose of this test is to compute the maximum possible data rate at which the switch can operate. These ideal conditions are defined as when the message traffic is one-way along a link and takes the shortest path through the switch from a source (src) to a sink (snk).

The generator transmitted 5000 1002-byte packets size at constant packet delay intervals. The maximum throughput was calculated by decreasing the packet transmitting delay on successive runs of the test. The maximum throughput recorded from the test was $\lambda = 0.531\ Mbps$ and the throughput efficiency (relative to the maximum link throughput) was $\lambda_{eff} = 75.06\%$.

Test 2: Congested External Link The worst possible performance of the packet routing switch is expected to take place when all messages entering the switch are bound for the same external link. Congestion may result at the external link TRAM, forcing the whole Transputer network to slow down. This condition is simulated by routing the traffic from three sources to the same sink.

The generator transmitted 5000 1002-byte packets at constant packet delay intervals. The maximum throughput was calculated by decreasing the packet transmitting delay on successive runs of the test. The maximum throughput recorded from the test was $\lambda = 0.568\ Mbps$ and the throughput efficiency was $\lambda_{eff} = 80.33\%$. The originally proposed reason for the increased throughput, in lieu of the expected decrease, is that the transputer's link utilization beomes somewhat more efficient under heavier loads. Based on subsequent tests of the enhanced design, the proposed cause is limited parallelism in the communication channels provided by the design.

Test 3: Impact of Dynamic Routing Table Updates The most interesting feature of the packet routing switch is its dynamic routing table update capability. The effects of dynamically loading routing table updates into the Transputer network during standard operation are of particular concern. Since the throughput will be directly affected by these updates, this test was performed to measure the throughput

as the update frequency was increased. For the sake of clarity the traffic patterns are kept constant in the switch while the throughput was recorded .

To analyze the effects of the routing table update injection the maximum throughput of the switch was first recorded prior to introducing the updates. The source transmitted 5000 1002–byte packets at constant packet delay intervals. The maximum throughput was calculated by decreasing the packet transmitting delay on successive runs of the test. For packets routed through the bridge, the maximum throughput recorded was $\lambda = 0.406\ Mbps$ and the throughput efficiency was $\lambda_{eff} = 57.40\%$. This contrasts with packets not routed through the bridge ($\lambda = 0.407\ Mbps$ and $\lambda_{eff} = 57.54\%$ respectively). The bridge used in this project provides only the service access point for host data communications onto the router network. Protocol conversions are not performed. Additional information on the bridges functionality is presented in Section 4.

Once the maximum operating throughput of the switch was recorded, the switch was driven at this constant throughput and routing table updates were introduced. The routing table updates used did not reflect changes of external link connected nodes, rather they reinitialized the external routing tables to their original value so that the throughput deterioration obtained could be compared to the maximum throughput obtained without the routing table updates. This deterioration demonstrates the overhead required to update external routing tables throughout the switch.

The update source used in the test was similar to the packet generators in capability. The only difference was that it distributed routing table update IDUs from the host. The update source sent five 12–byte routing table update IDUs, each destined for a separate TRAM in the network. These update IDUs were delivered one after another without using any inter-departure delays. Once all five updates were injected into the Transputer network the update source waited for the inter-transmission delay and then repeated the same operation. Updates were generated throughout the duration of the test.

The same source characteristics were used to drive the switch again, except both sources were driven at the maximum allowable throughputs, as recorded earlier. The maximum throughputs recorded from the

test were the same as without routing updates and the effect on the corresponding throughput efficiencies was considered negligible.

Tests 4 and 5: General Switch Operation The last two tests were conducted to generate data which would reflect the packet routing switch's general operational characteristics. In these cases, random destination traffic would be passing through the switch with varying packet sizes and varying inter-arrival times. Since we are interested in the switch's general operation in the project testbed, packet sizes and inter-arrival times were matched to the expected traffic in the testbed.

Test 4 was designed to analyze throughput of the switch with random destination packets passing through the switch. Sources and sinks were connected to all the external links. An additional sink was included to absorb packets with destinations set for the packet routing switch's host node.

The generator used a normally distributed packet size and a normally distributed packet delay with a uniformly distributed random packet destination field. The maximum aggregate throughput was calculated for each sink by decreasing the packet transmitting delay on successive runs of the test. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{SINK } 0 \ \lambda = 0.121 \ Mbps \\ &\text{SINK } 1 \ \lambda = 0.116 \ Mbps \\ &\text{SINK } 2 \ \lambda = 0.110 \ Mbps \\ &\text{SINK } 3 \ \lambda = 0.118 \ Mbps \\ &\text{SINK } 4 \ \lambda = 0.116 \ Mbps \end{aligned} \tag{1}$$

To analyze the general operation of the switch, random traffic patterns were used in Test 5. The configuration used for Test 5 was identical to Test 4 except an additional source was included to model traffic originating from the host node.

The generator used a normally distributed packet size and a normally distributed packet delay with a uniformly distributed random packet destination field . The maximum aggregate throughput was

calculated for each sink by decreasing the packet transmitting delay on successive runs of the test. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{SINK 0} \ \lambda = 0.068\ Mbps \\ &\text{SINK 1} \ \lambda = 0.063\ Mbps \\ &\text{SINK 2} \ \lambda = 0.066\ Mbps \\ &\text{SINK 3} \ \lambda = 0.064\ Mbps \\ &\text{SINK 4} \ \lambda = 0.066\ Mbps \end{aligned} \tag{2}$$

Table 1 provides a comprehensive listing of the results obtained from the tests performed. $\lambda_{aggregate}$ in Table 1 corresponds to the sum of all the maximum throughputs achieved at all sinks during a test. This metric provides a valuable insight into the functioning of the packet routing switch when it is viewed as a black box device. By comparing the aggregate throughput for different test cases, valid comparison for the throughput of the switch can be made.

Test	$\lambda_{aggregate}$		λ	$\lambda_{relative}$
Transputer Link Throughput	0.707 Mbps		λ_{max} = 0.707 Mbps	N/A
1	0.531 Mbps		λ = 0.531 Mbps	λ_{eff} = 75.06%
2	0.568 Mbps		λ = 0.568 Mbps	λ_{eff} = 80.33%
3	0.812 Mbps	Direct Traffic (no updates)	λ = 0.407 Mbps	λ_{eff} = 57.54%
		Direct Traffic (w/ updates)	λ = 0.407 Mbps	λ_{del} = 0%
		Through Bridge (no updates)	λ = 0.406 Mbps	λ_{eff} = 57.40%
		Through Bridge (w/ updates)	λ = 0.406 Mbps	λ_{del} = 0%
Test 4	0.581 Mbps	Sink 0	λ = 0.121 Mbps	N/A
		Sink 1	λ = 0.116 Mbps	N/A
		Sink 2	λ = 0.110 Mbps	N/A
		Sink 3	λ = 0.118 Mbps	N/A
		Sink 4	λ = 0.116 Mbps	N/A
Test 5	0.327 Mbps	Sink 0	λ = 0.068 Mbps	N/A
		Sink 1	λ = 0.063 Mbps	N/A
		Sink 2	λ = 0.066 Mbps	N/A
		Sink 3	λ = 0.064 Mbps	N/A
		Sink 4	λ = 0.066 Mbps	N/A

Table 1 Cumulative results

3 TRANSPUTER NETWORK DESIGN

The new design is illustrated in Figure 6. This black box design provides for optimal routing of data traffic. The required high-speed channels are provided for the host link and each of the four external (source/sink) links. A separate router is dedicated to each external link to provide optimal handling of arriving message traffic. Furthermore, two router/driver pairs are placed on a single Transputer site to optimize communications using the faster hard links. Routers are connected to provide fully parallel channels of communication between each distinct pair of routers.

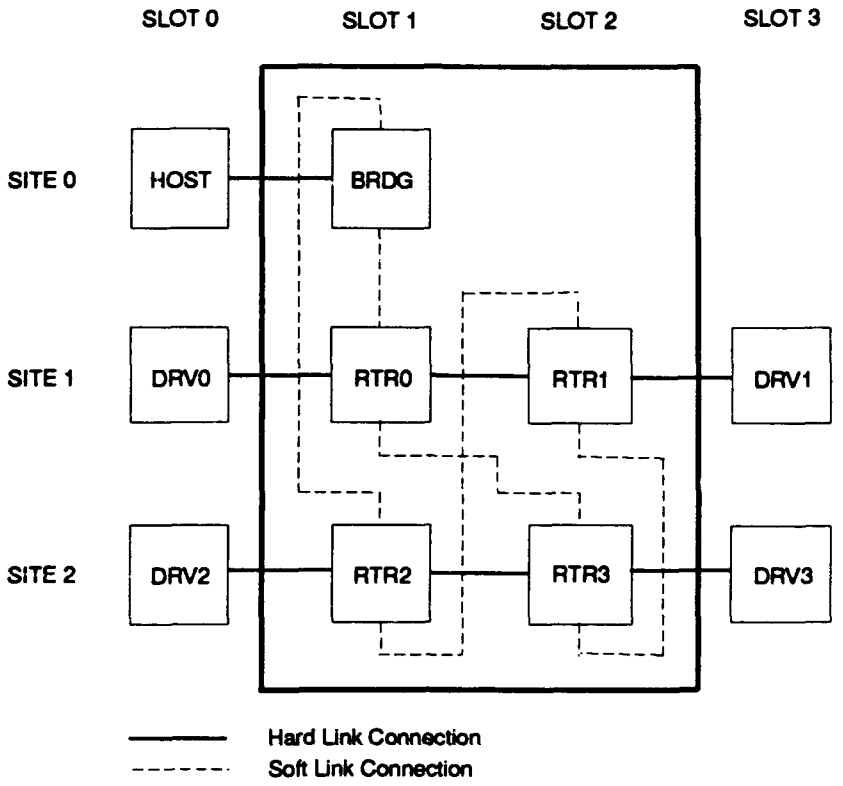


Figure 6 Transputer architecture.

To provide a clearer understanding of the overall models functionality, the corresponding software process names are shown on their respectively assigned transputer in Figure 6. This also ensures that the software and hardware configurations match. Because communication channels between processes on separate transputers are mapped directly onto hardware links by the Transputer boot loader, the software and hardware configurations must match to prevent a load failure. The mapping specifications are declared in an accompanying OCCONF configuration program to provide for the parallel execution of the host, bridge, router, and driver processes on their respective assigned processors.

4 PROCESS DESIGN

Preliminary Implementation

The *Communicating Sequential Processes* (CSP) (Hoare 1985) specification methodology provides the necessary means for specifying concurrent occam processes. It also supports the refinement of

specifications through multiple levels of abstraction to enhance reasoning about inter-process behavior. This understanding helps to ensure that the correct functionality is achieved and provides confidence that deadlock will not occur. In addition, well-defined CSP specifications are readily refined into occam code which can subsequently be executed on a Transputer system with minimal debugging. The occam processes described in the following paragraphs were directly refined from the CSP specifications provided in Appendix A.

The host process drives the host communications link, as shown in Figure 7. In accordance with occam programming conventions, it serves as the single service access point for all host PC I/O. Its primary function is to introduce data packets and routing table updates. It is also responsible for overall control of process execution to provide for the synchronization of the driver processes at each stage during a test. A single stage for example, might record the throughput of 5000 1002-byte data packets with a mean inter-packet arrival time of .010 seconds. After re-synchronization, the next stage would begin with packet transmissions distributed with a mean arrival time of .009 seconds, etc. Occam discriminated channel protocol signals are used to provide re-synchronization at each new stage and the clean termination of all parallel processes upon test completion.

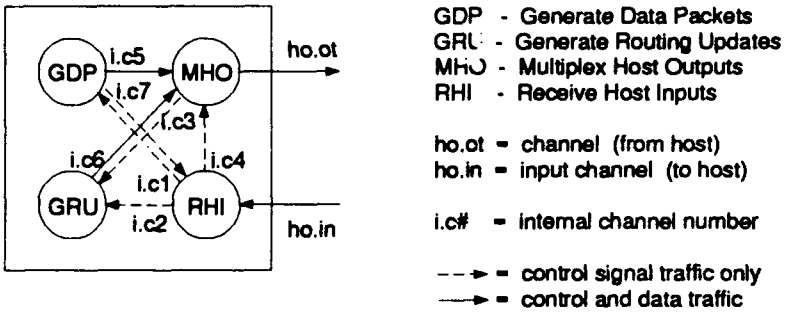


Figure 7 Host Communications Flow Diagram

The bridge process serves as the access point for the placement and extraction of host traffic to and from the router communications network, as shown in Figure 8. Other message traffic on the router network which enters the bridge is re-transmitted to the corresponding router on the opposite side of the bridge. Control signals on the router network which enter the bridge are captured and forwarded

to the host process. Control signals from the host are transmitted to both routers directly connected to the bridge. These routers are subsequently responsible for forwarding appropriate host signals to their assigned partners. For performance reasons, each of the two routers not directly connected to the bridge were coupled with the directly connected router sharing the same Transputer site.

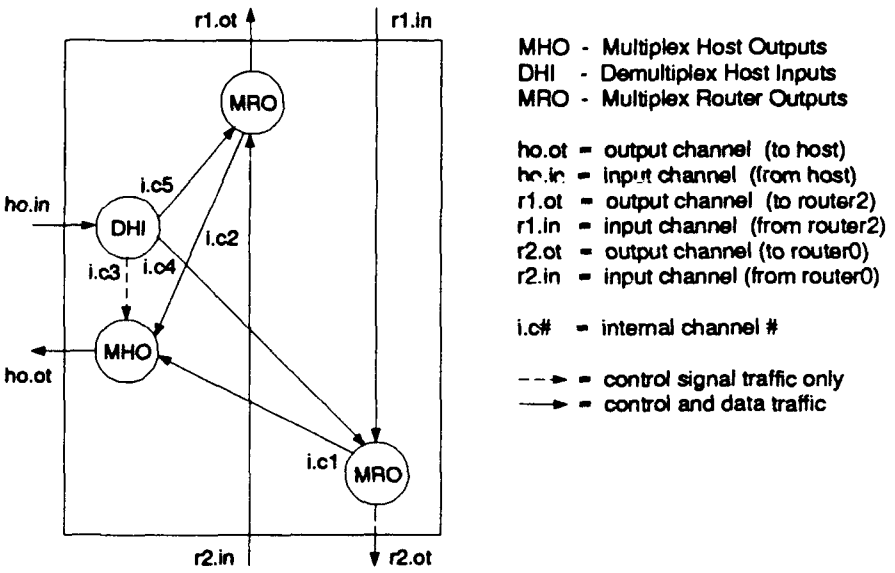


Figure 8 Bridge Communications Flow Diagram

Router processes, as shown in Figure 9, direct message traffic flow around the router network. Since fully connected routers are used, every external data message must pass through exactly two routers. This greatly simplifies the routing process since any external data message arriving on any channel from another router can be directly transferred to its associated driver (sink) process. Routing table lookups are only required when an external message is received from the driver (source) or an internal message is received from the host (source) to ensure the message is forwarded to the appropriate destination router.

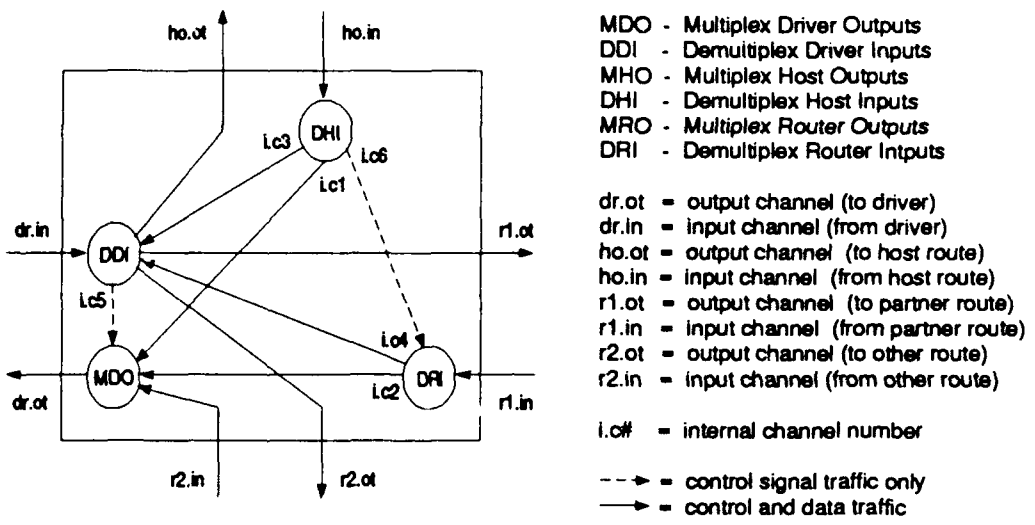


Figure 9 Router Communications Flow Diagram

Drivers perform two primary functions, as shown in Figure 10. First, they serve as the source for data message traffic on the router network. Second, they act as a sink for all message traffic arriving from the router network. Inter-packet arrival data is accumulated for post-test result calculations.

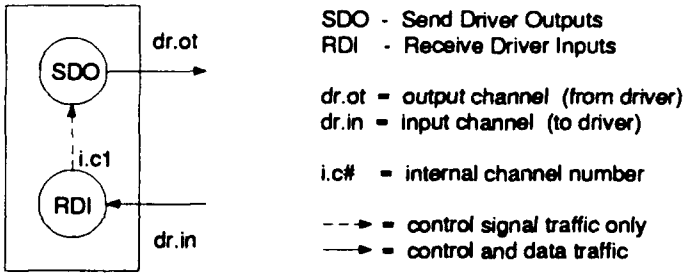


Figure 10 Driver Communications Flow Diagram

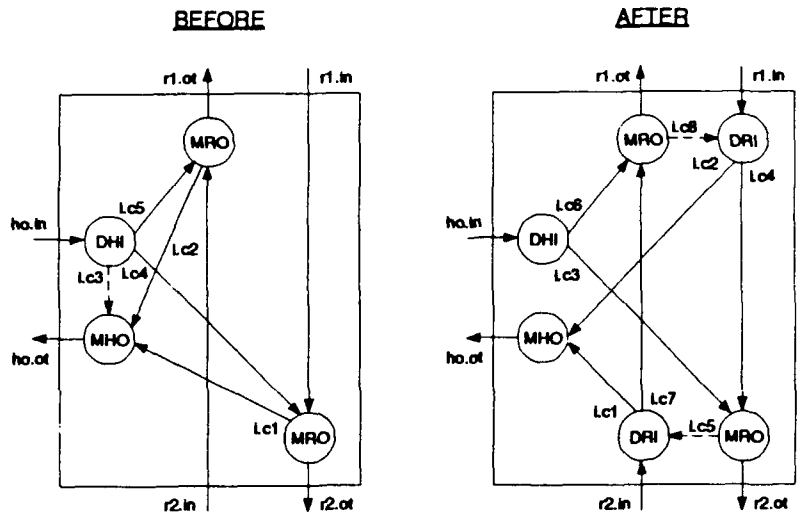
To eliminate any potential impact on test results, all distributed values were calculated in advance and stored in arrays for easy retrieval during the appropriate test stage. External link drivers, serving in their capacity as sinks, also store only inter-packet arrival results. All results are retrieved and calculations are performed at the completion of each stage of message routing tests. No host PC I/O is performed during the testing process. These measures ensure the accuracy of routing throughput test results.

Enhanced Implementation

Results from the preliminary design tests were promising. The throughput speeds achieved provided conclusive proof of the T800's suitability for the project testbed. However, two observations from the preliminary design tests provided reasons for concern. The first observation was encountered during the second part of test P3. During this test, a livelock condition occurred when the arrival times between successive routing update IDUs dropped below 30 ticks (or CPU clock cycles). The other observation occurred during test P5 when the system became deadlocked as inter-packet arrival times dropped below 50 ticks.

The routing update problem was located within the multiplex host output (MHO) process shown in Figure 7. The occam ALT construct is used in this process, so that when a single channel is ready to transmit, that channel is served, while if more than one channel is ready to transmit, the ALT indiscriminately chooses which one to serve. The implementation of the ALT statement checks the channels in sequential order as listed within the ALT statement body, with each new pass starting at the top of the list. Thus, when the routing update channel was listed first and updates arrived at a high enough rate, the updates consumed all the time slices so that no other communications were serviced. The livelock condition was eliminated by modifying the channel list so that the routing update channel appeared last.

The deadlock situation encountered in test P5 was more serious. By examining the preliminary CSP specification and the steps in the refinement process, it became apparent that measures taken to reduce process contention and packet duplication were at the root of the problem. These measures were designed to reduce internal communications to an absolute minimum. To accomplish this, ALT processes were used to multiplex appropriate input channels onto the respective output channels within the bridge and router processes, as shown in the "before" sections of Figures 11 and 12. Only the host and driver input and output channels were separated into distinct parallel processes.



MHO - Multiplex Host Outputs
DHI - Demultiplex Host Inputs
MRO - Multiplex Router Outputs
DRI - Demultiplex Router Inputs

--► = control signal traffic only
——► = control and data traffic

ho.ot = output channel (tc host)
ho.in = input channel (from host)
r1.ot = output channel (to router2)
r1.in = input channel (from router2)
r2.ot = output channel (to router0)
r2.in = input channel (from router0)

i.c# = internal channel #

Figure 11 Bridge Communications Flow Enhancements

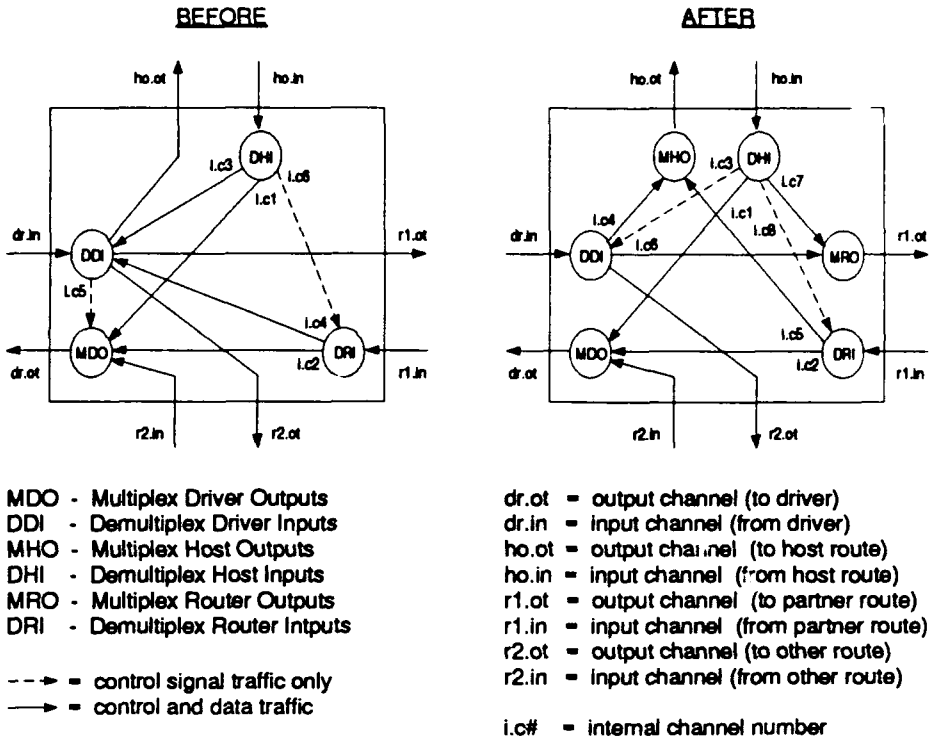


Figure 12 Router Communications Flow Enhancements

The problem with this approach is that the parallel lines of communication at the bridge and routers were coupled together too tightly. At the bridge, traffic between connected routers was coupled to the host's outbound traffic. And, at router 0 and router 2, host communications to driver 1 and driver 3 were coupled to driver 0 and driver 2 communications, respectively. Deadlock could occur if four synchronized data packets were transmitted; from the host to driver 1 and driver 3, from driver 0 to driver 2, and from driver 2 to driver 0. Under these circumstances, driver 0 and driver 2 deadlocked because the bridge had blocked to serve the host communications. The host deadlocked because router 0 and router 2 had blocked to serve the driver 0 and driver 2 communications, respectively. Eventually, the remainder of the router network filled up with blocked traffic, leading to complete system deadlock.

To solve the deadlock problem, all inbound and outbound channels at the bridge and routers were decoupled as shown in the "after" sections of Figures 11 and 12 for the enhanced version of the satellite model. During the refinement process, an interesting observation was made. The effect of the refinements

was to restrict each ALT process in the model to either a single input or a single output channel for data traffic communications, eliminating the deadlock condition, but also, by opening up parallel channels of communication, significantly increasing all system throughput rates.

5 SIMULATION RESULTS

Preliminary Implementation

Test P0: Maximum Link Throughput To ascertain the raw data rate of the configured system, the throughput of a single Transputer link was measured. The measured value was calculated by configuring the Host (sink) and Driver1 (source) processes on a directly connected hard link. The source used a constant 1002-byte packet size and a constant packet delay. The maximum throughput was calculated by decreasing the packet transmitting delay on successive runs of the test. The maximum throughput recorded from the test was $\lambda_{max} = 13.464\ Mbps$.

Test P1: Maximum Switch Throughput The first test involves recording the throughput of the packet routing switch under ideal conditions. The maximum throughput recorded from the test was $\lambda = 6.811\ Mbps$ and the throughput efficiency (relative to the maximum link throughput) was $\lambda_{eff} = 50.59\%$.

Test P2: Congested External Link The worst possible performance of the packet routing switch is conceived as when all messages entering the switch are bound for the same external link. Congestion may result at the external link TRAM forcing the whole Transputer network to slow down. The maximum throughput recorded from the test was $\lambda = 7.967\ Mbps$ and the throughput efficiency was $\lambda_{eff} = 59.17\%$.

Test P3: Impact of Dynamic Routing Table Updates To analyze the effects of the routing table update injection, the maximum throughput of the switch was first recorded prior to introducing the updates. For packets routed through the bridge, the maximum throughput recorded was $\lambda = 3.700\ Mbps$

and the throughput efficiency was $\lambda_{eff} = 27.48\%$. This contrasts with packets not routed through the bridge ($\lambda = 4.800\ Mbps$ and $\lambda_{eff} = 35.65\%$ respectively).

Once the maximum operating throughput of the switch was recorded, the switch was driven at this constant throughput and routing table updates were introduced. This test demonstrates the overhead required to update external routing tables throughout the switch. For packets passing through the bridge, the maximum throughput recorded from the test was $\lambda = 3.504\ Mbps$ and the throughput efficiency was $\lambda_{det} = 5.30\%$. This contrasts with packets not routed through the bridge ($\lambda = 4.740\ Mbps$ and $\lambda_{det} = 1.25\%$ respectively).

Tests P4 and P5: General Switch Operation The last two tests were conducted to generate data which would reflect the packet routing switch's general operational characteristics. Figure 13 shows a plot of the throughput versus the packet transmission delay. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{HOST} && \lambda = 3.913\ Mbps \\ &\text{DRIVER 0} && \lambda = 2.776\ Mbps \\ &\text{DRIVER 1} && \lambda = 2.776\ Mbps \\ &\text{DRIVER 2} && \lambda = 2.806\ Mbps \\ &\text{DRIVER 3} && \lambda = 2.843\ Mbps \end{aligned} \tag{3}$$

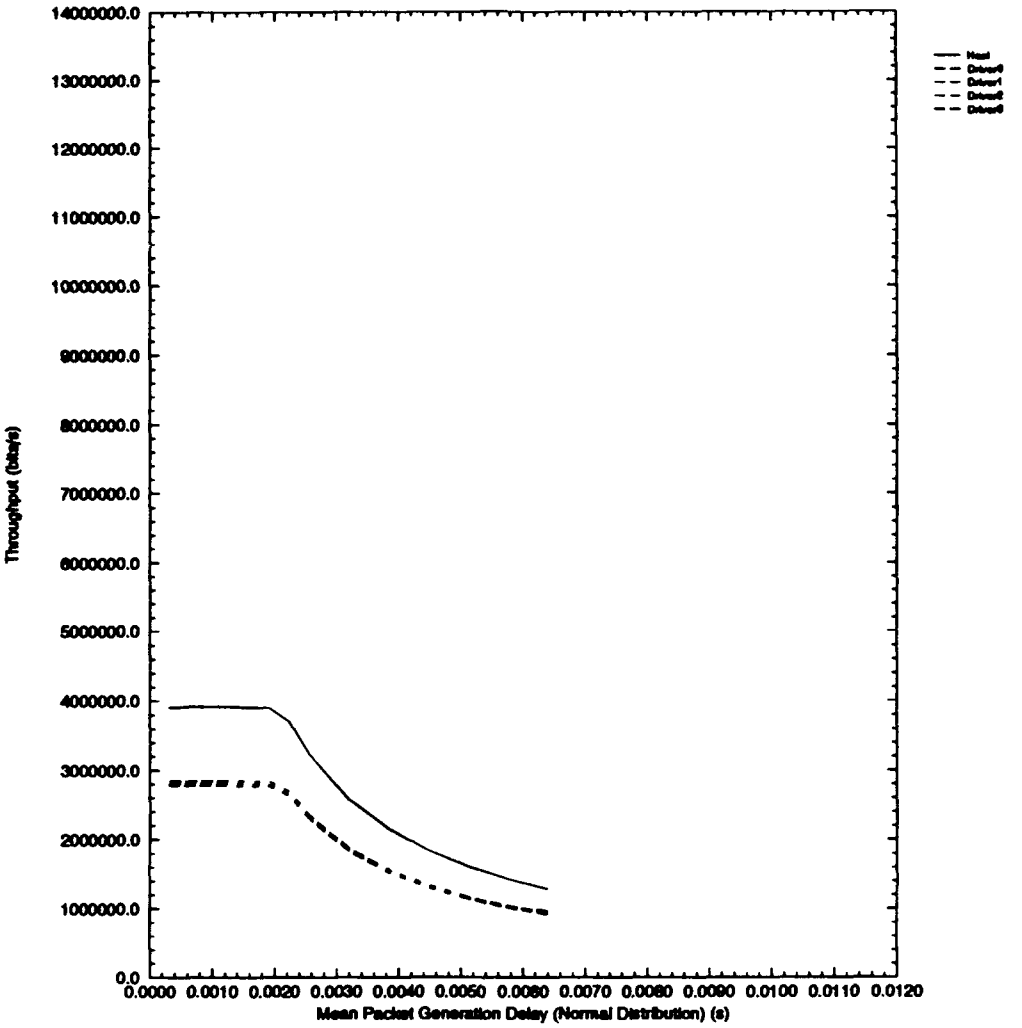


Figure 13 Test P4: Switch general operation throughput plot. (Preliminary Design)

The configuration used for Test P5 was identical to Test P4 with the inclusion of message traffic from the host. Figure 14 shows a plot of the throughput versus the packet transmission delay. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{HOST} && \lambda = 2.152\ Mbps \\ &\text{DRIVER 0} && \lambda = 2.169\ Mbps \\ &\text{DRIVER 1} && \lambda = 2.063\ Mbps \\ &\text{DRIVER 2} && \lambda = 2.049\ Mbps \\ &\text{DRIVER 3} && \lambda = 2.091\ Mbps \end{aligned} \tag{4}$$

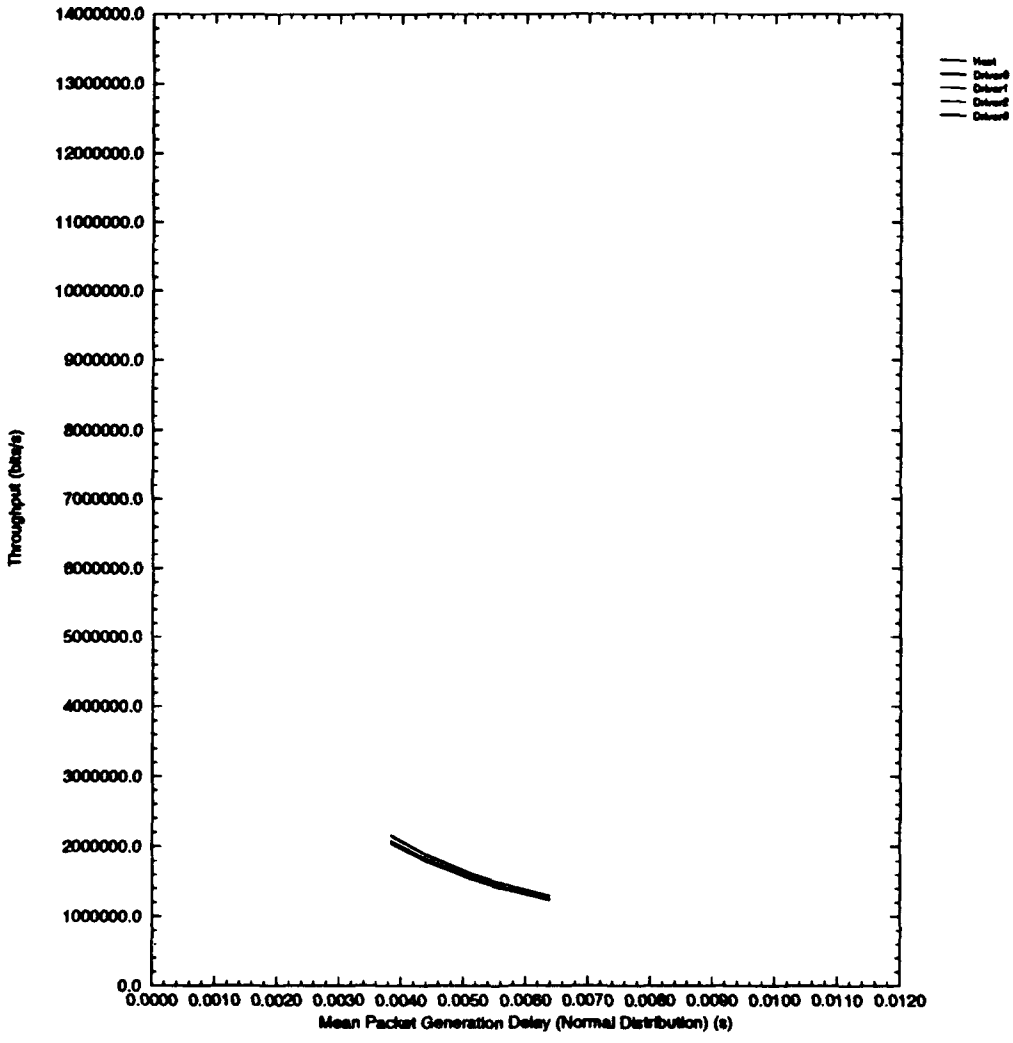


Figure 14 Test P5: Switch general operation throughput plot. (Preliminary Design)

The results obtained from this simulation are summarized in Table 2 which includes the aggregate data rate for the routing switch taken as a whole.

Test	$\lambda_{aggregate}$		λ	$\lambda_{relative}$
Transputer Link Throughput	13.464 Mbps		λ_{max} = 13.464 Mbps	N/A
1	6.811 Mbps		λ = 6.811 Mbps	λ_{eff} = 50.59%
2	7.967 Mbps		λ = 7.967 Mbps	λ_{eff} = 59.17%
3	8.500 Mbps	Direct Traffic (no updates)	λ = 4.800 Mbps	λ_{eff} = 35.65%
		Direct Traffic (w/ updates)	λ = 4.740 Mbps	λ_{del} = 1.25%
		Through Bridge (no updates)	λ = 3.700 Mbps	λ_{eff} = 27.48%
		Through Bridge (w/ updates)	λ = 3.504 Mbps	λ_{del} = 5.30%
4	15.114 Mbps	Host	λ = 3.913 Mbps	N/A
		Driver 0	λ = 2.776 Mbps	N/A
		Driver 1	λ = 2.776 Mbps	N/A
		Driver 2	λ = 2.806 Mbps	N/A
		Driver 3	λ = 2.843 Mbps	N/A
5	10.524 Mbps	Host	λ = 2.152 Mbps	N/A
		Driver 0	λ = 2.169 Mbps	N/A
		Driver 1	λ = 2.063 Mbps	N/A
		Driver 2	λ = 2.049 Mbps	N/A
		Driver 3	λ = 2.091 Mbps	N/A

Table 2 Cumulative results

Enhanced Implementation

Test E0: Maximum Link Throughput To ascertain the raw data rate of the enhanced system, the throughput of a single Transputer link was again measured by configuring the Host (sink) and Driver1 (source) processes on a directly connected hard link. Figure 15 shows a plot of the throughput versus the packet transmission delay. The maximum throughput recorded from the test was λ_{max}= 13.468 $Mbps$, a slight improvement of about 0.004 Mbps over the preliminary implementation.

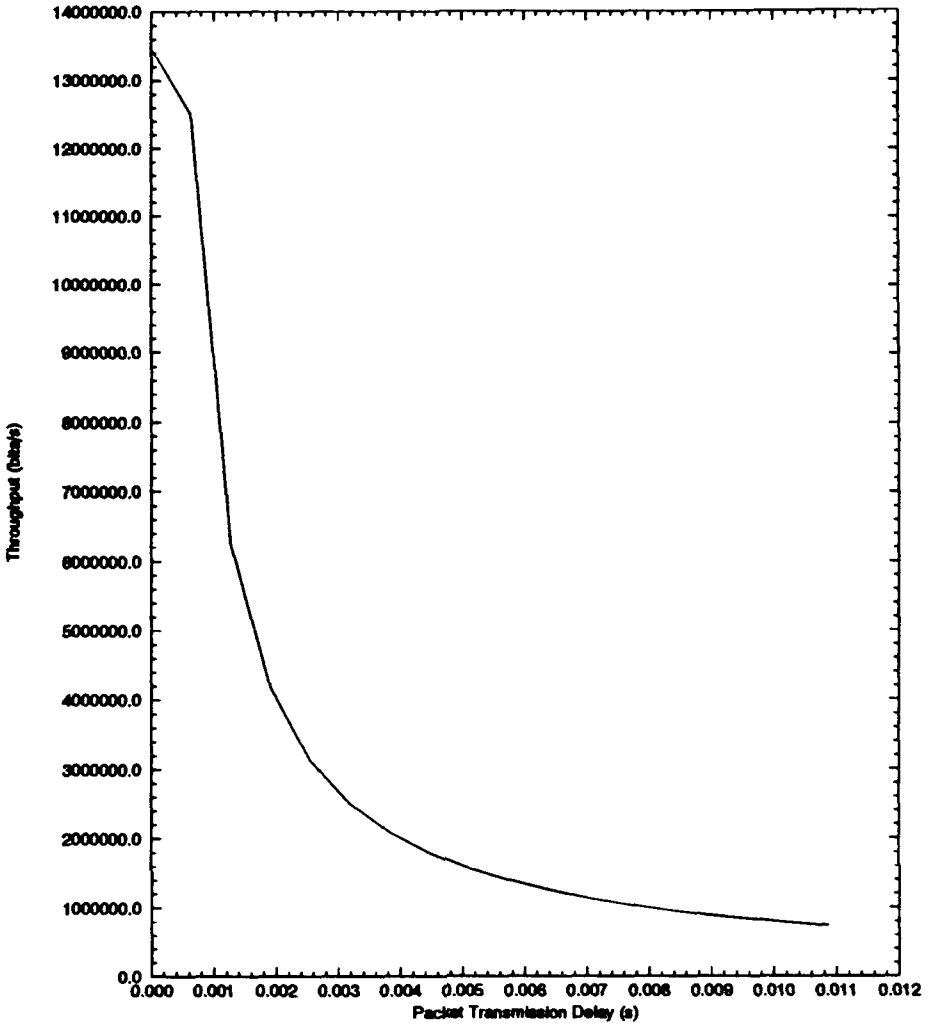


Figure 15 Test E0: TRAM Link throughput plot.

Test E1: Maximum Switch Throughput The first test involves recording the throughput of the packet routing switch under ideal conditions. Figure 16 shows a plot of the throughput versus the packet transmission delay. The maximum throughput recorded from the test was $\lambda = 11.454\ Mbps$ and the throughput efficiency (relative to the maximum link throughput) was $\lambda_{eff} = 85.05\%$. This is slightly more than a 35% improvement over the preliminary implementation.

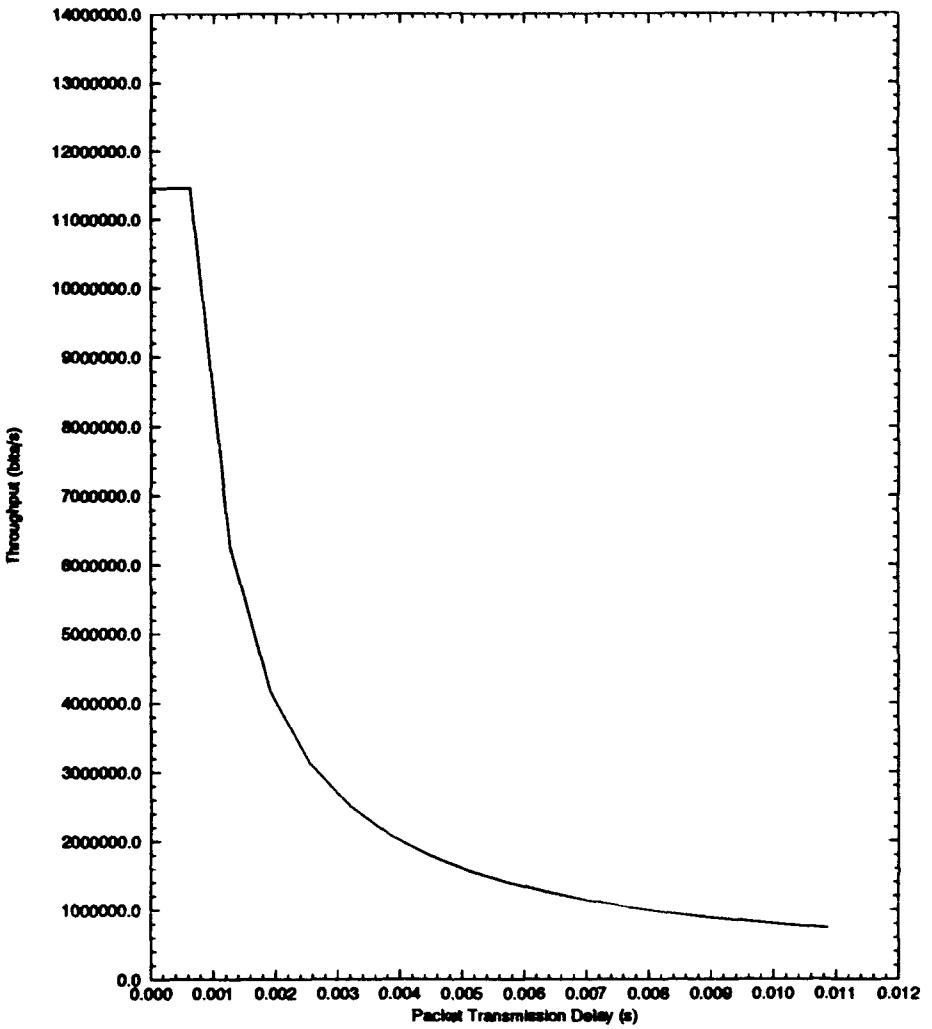


Figure 16 Test E1: Maximum switch throughput plot.

Test E2: Congested External Link The worst possible performance of the packet routing switch is conceived as when all messages entering the switch are bound for the same external link. Congestion may result at the external link TRAM forcing the whole Transputer network to slow down. Figure 17 shows a plot of the throughput versus the packet transmission delay. The maximum throughput recorded from the test was $\lambda = 7.856\ Mbps$ and the throughput efficiency was $\lambda_{eff} = 58.33\%$, a decrease of about .110 Mbps and less than 1% from the preliminary implementation.

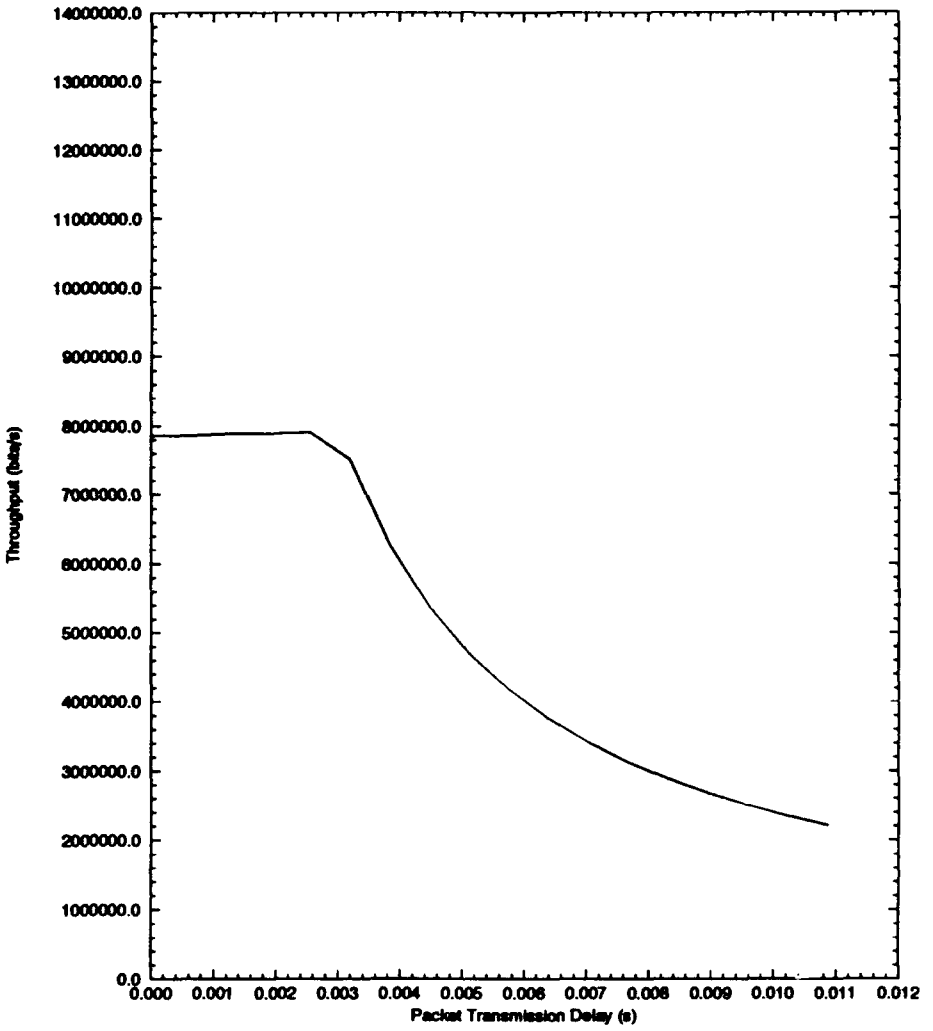


Figure 17 Test E2: Congested external link throughput plot.

Test E3: Impact of Dynamic Routing Table Updates To analyze the effects of the routing table update injection, the maximum throughput of the switch was first recorded prior to introducing the updates. Figure 18 shows a plot of the throughput versus the packet transmission delay. For packets routed through the bridge, the maximum throughput recorded was $\lambda = 4.789\ Mbps$ and the throughput efficiency was $\lambda_{eff} = 35.56\%$, an increase of 1.089 Mbps and over 8% respectively from the preliminary implementation. This contrasts with packets not routed through the bridge ($\lambda = 6.678\ Mbps$ and $\lambda_{eff} = 49.58\%$ respectively), an increase of 1.877 Mbps and over 14% respectively from the preliminary

implementation.

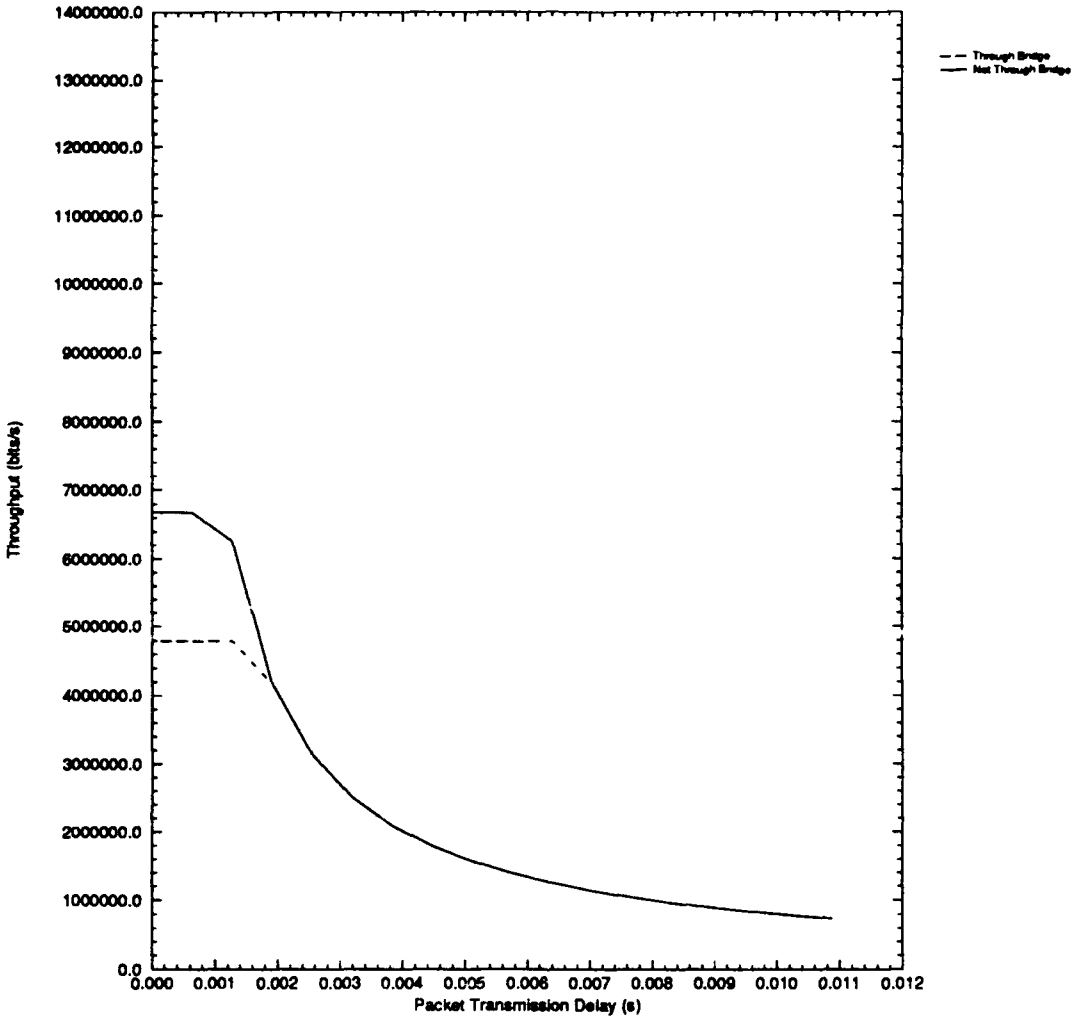


Figure 18 Test E3: Normal link throughput plot for cross-traffic.

Once the maximum operating throughput of the switch was recorded, the switch was driven at this constant throughput and routing table updates were introduced. This test demonstrates the overhead required to update external routing tables throughout the switch. Figure 19 shows a plot of the throughput versus the update transmission delay. For packets passing through the bridge, the maximum throughput recorded from the test was $\lambda = 4.711\ Mbps$ and the throughput cfficiency deteriorated by $\lambda_{det} = 1.63\%$. This contrasts with packets not routed through the bridge ($\lambda = 4.710\ Mbps$ and $\lambda_{det} = 29.47\%$ respectively).

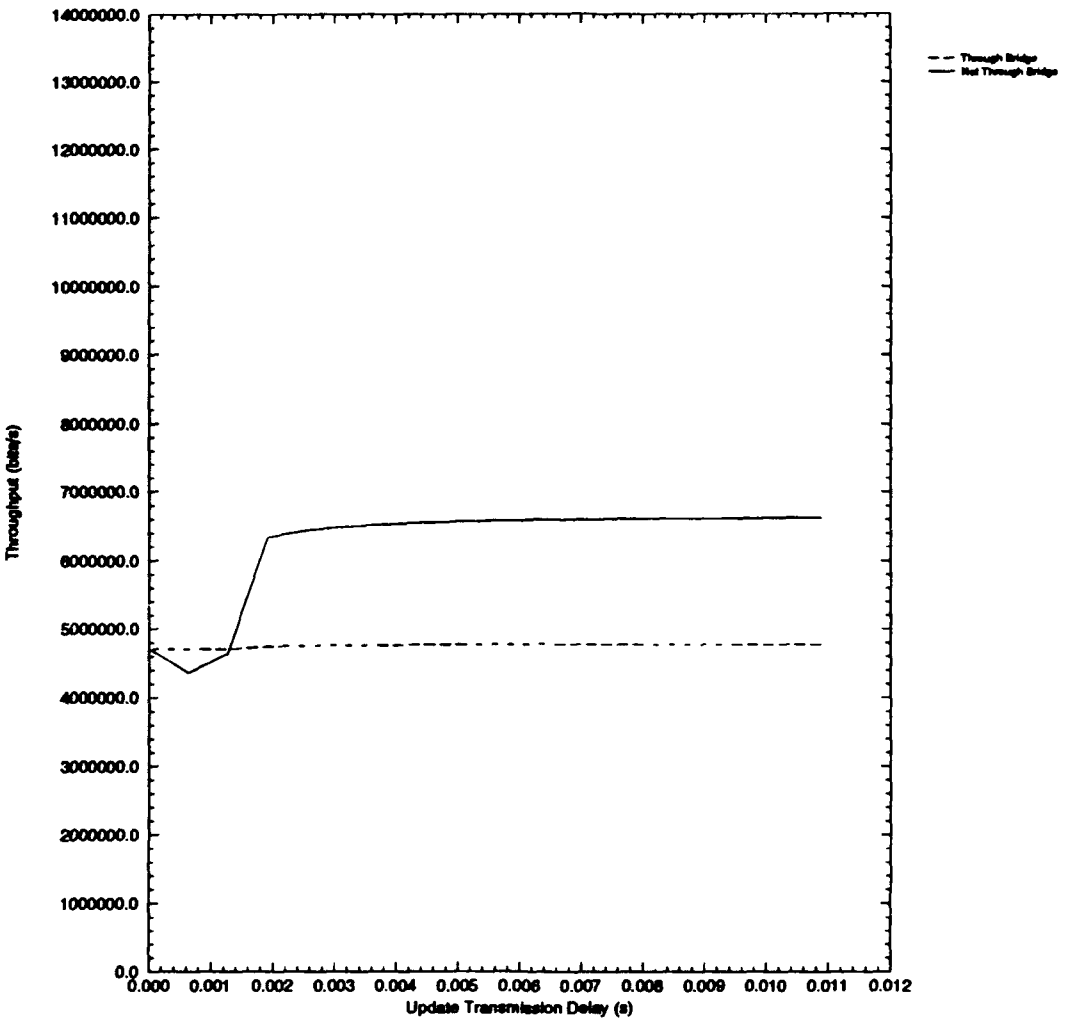


Figure 19 Test E3: Throughput w/ Increasing Routing Table Update Frequency.

Tests E4 and E5: General Switch Operation The last two tests were conducted to generate data which would reflect the packet routing switch's general operational characteristics. Figure 20 shows a plot of the throughput versus the packet transmission delay. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{HOST} && \lambda = 5.746\ Mbps \\ &\text{DRIVER 0} && \lambda = 4.349\ Mbps \\ &\text{DRIVER 1} && \lambda = 4.092\ Mbps \\ &\text{DRIVER 2} && \lambda = 4.127\ Mbps \\ &\text{DRIVER 3} && \lambda = 4.184\ Mbps \end{aligned} \tag{5}$$

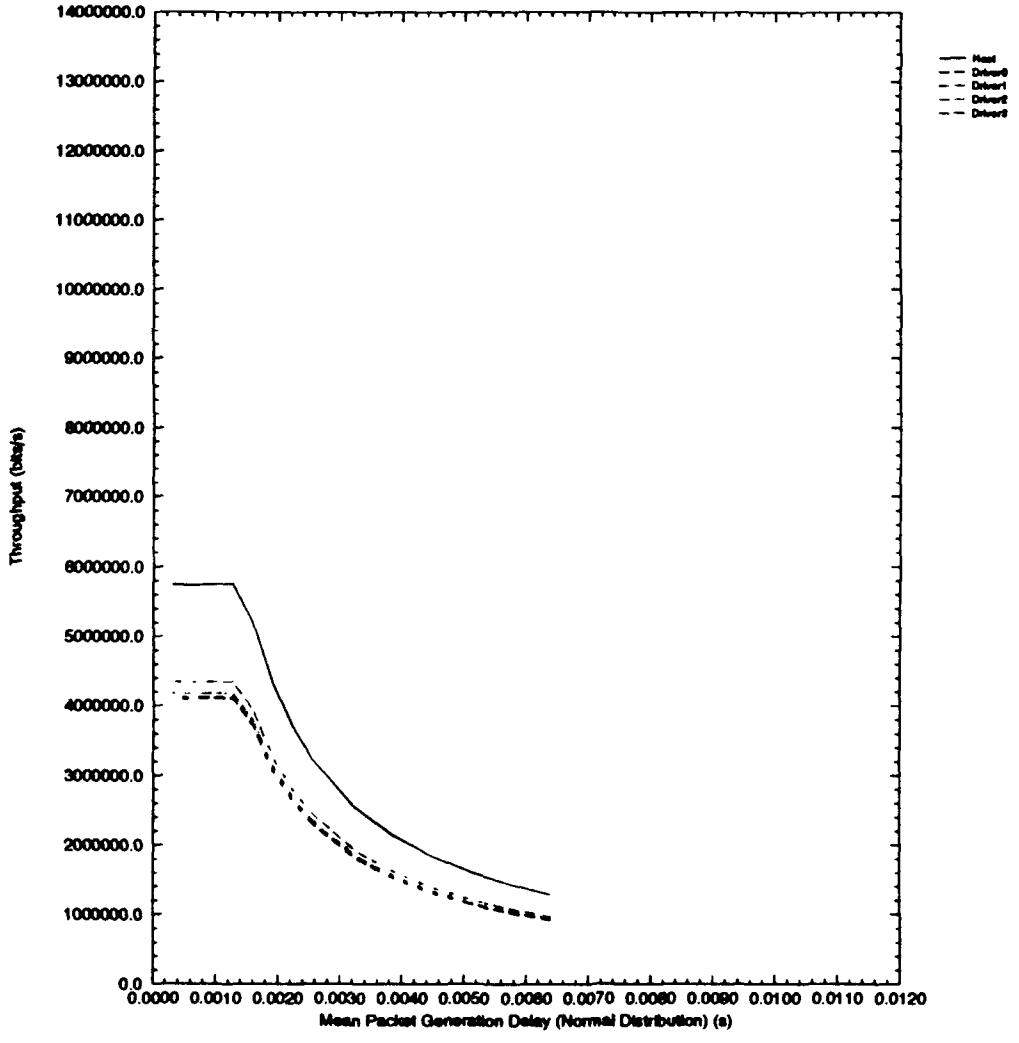


Figure 20 Test E4: Switch general operation throughput plot.

The configuration used for Test E5 was identical to Test E4 with the inclusion of message traffic from the host. Figure 21 shows a plot of the throughput versus the packet transmission delay. The maximum throughputs recorded from the test were:

$$\begin{aligned} &\text{HOST} && \lambda = 4.651 \\ &\text{DRIVER 0} && \lambda = 4.690\ Mbps \\ &\text{DRIVER 1} && \lambda = 4.457\ Mbps \\ &\text{DRIVER 2} && \lambda = 4.427\ Mbps \\ &\text{DRIVER 3} && \lambda = 4.698\ Mbps \end{aligned} \tag{6}$$

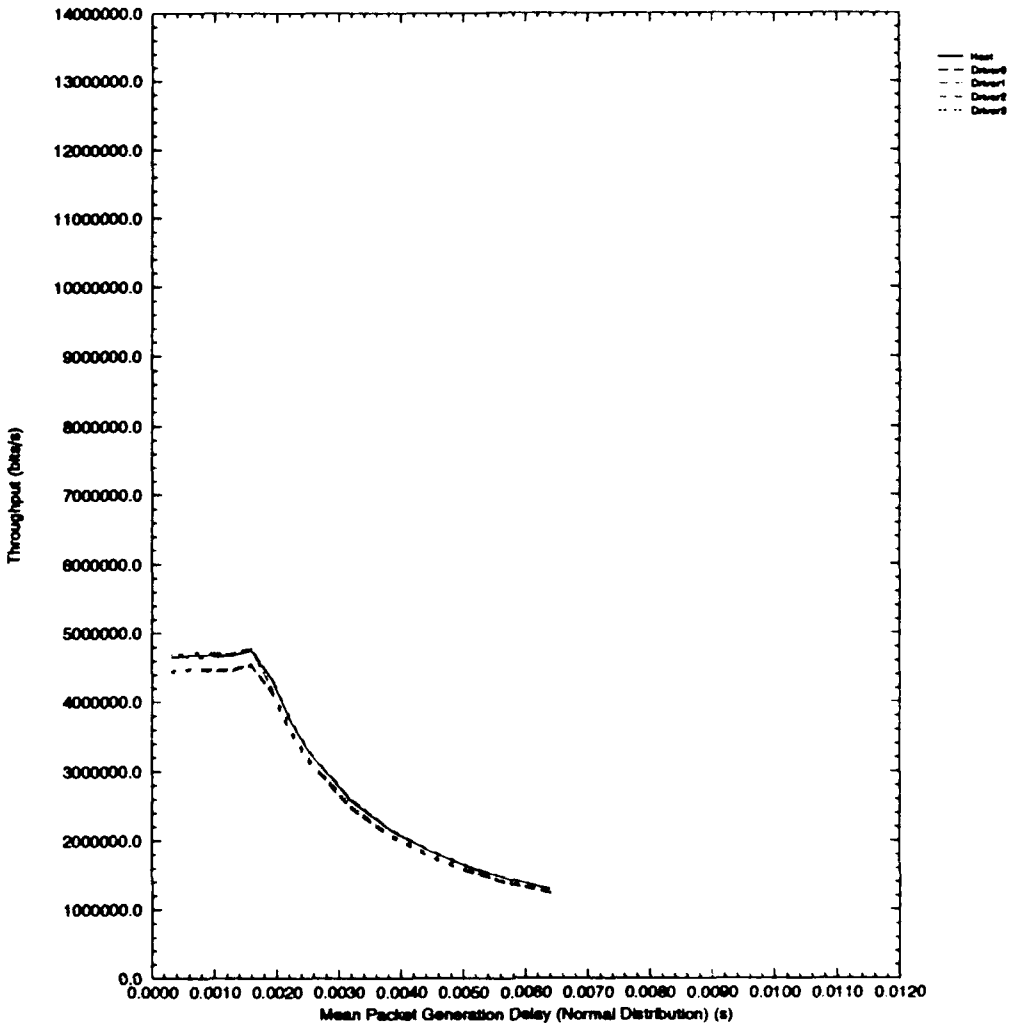


Figure 21 Test E5: Switch general operation throughput plot.

The results obtained from this simulation are summarized in Table 3 which includes the aggregate data rate for the routing switch taken as a whole.

Test	$\lambda_{aggregate}$		λ	$\lambda_{relative}$
Transputer Link Throughput	13.468 Mbps		λ_{max} = 13.468 Mbps	N/A
1	11.454 Mbps		λ = 11.454 Mbps	λ_{eff} = 85.05%
2	7.856 Mbps		λ = 7.856 Mbps	λ_{eff} = 58.33%
3	11.467 Mbps	Direct Traffic (no updates)	λ = 6.678 Mbps	λ_{eff} = 49.58%
		Direct Traffic (w/ updates)	λ = 4.710 Mbps	λ_{det} = 29.47%
		Through Bridge (no updates)	λ = 4.789 Mbps	λ_{eff} = 35.56%
		Through Bridge (w/ updates)	λ = 4.711 Mbps	λ_{det} = 1.63%
4	22.498 Mbps	Host	λ = 5.746 Mbps	N/A
		Driver 0	λ = 4.349 Mbps	N/A
		Driver 1	λ = 4.092 Mbps	N/A
		Driver 2	λ = 4.127 Mbps	N/A
		Driver 3	λ = 4.184 Mbps	N/A
5	22.923 Mbits/s	Host	λ = 4.651 Mbps	N/A
		Driver 0	λ = 4.690 Mbps	N/A
		Driver 1	λ = 4.457 Mbps	N/A
		Driver 2	λ = 4.427 Mbps	N/A
		Driver 3	λ = 4.698 Mbps	N/A

Table 3 Cumulative results

6 CONCLUSIONS

In this report, we have described the design process used to implement a Transputer based packet routing switch using cooperating OCCAM processes on a network of T800 Transputers.

The maximum Transputer link throughput calculated was 13.468 Mbps. This computed value is consistent with the Inmos published link speed of 20 Mbps with consideration for byte-transfer overhead bits. This also compares very favorably with the previous (Genesys and C based) benchmark of only

3.81 Mbps. The recorded throughputs are sufficient to warrant consideration of the Transputer system's suitability to the project testbed.

The maximum switch throughput recorded was 11.454 Mbps with a throughput efficiency of 85.05%. The throughput and throughput efficiency indicate that the software design functions *well* within the required testbed range.

One significant change from previous results was gathered during Test E2. In this test, the throughput recorded was 7.856 Mbps with a throughput efficiency of 58.33%. Contrary to results seen in the original and preliminary design tests, these throughput results are less than those of Test E1 (which mimics ideal switch routing conditions). The lower throughput in the enhanced design is due to congestion at the switch. The significant change between models was a decoupling of the communication channels to open parallel channels of communication. This result suggests that the effects of congestion become more prominent when greater concurrency is achieved in Transputer systems.

The change in the throughput of the switch traffic in Test E3 was considered minimal, since routing updates would not sustain such high rates in an actual system. The effects at normal update rates are considered negligible. This is not a surprising considering the fact that a routing message destined for each internal TRAM is only 5 bytes long (including interface control information). The processing required to update a node's external routing table is also minimal, since all addresses are stored as 2-byte values in an integer array.

The throughput results obtained after conducting Test E4 and Test E5 are comparable to those obtained from Test E2 (the congested external link test). This is expected since all external links of the packet routing switch are heavily loaded during Test E4 and Test E5, similar to the conditions encountered during Test E2. In Test E4, the differences can be attributed to the additional presence of traffic on the routers destined for other sinks. In Test E5, further differences can be attributed to the additional traffic load from the host which may require some rerouting and which competes directly with the respective drivers load entering the router network. The addition of a packet generating source at the host link connection

increases the aggregate throughput by 0.425 Mbps. The minimal driver throughput results of 4.092 Mbps (Test E4) and 4.427 Mbps (Test E5) are a significant improvement over the previous (Genesys and C based) design of only 0.155 Mbps and 0.064 Mbps respectively.

The data suggests that the design of the switch yields throughput results that are clearly within the operating range specified for the testbed nodes. Therefore, the suitability of this Transputer implementation for the project testbed is clear.

7 REFERENCES

Chow, Y.C; R.E. Newman-Wolfe; C. Ward and C.D. McLochlin. 1988. "A Simulation Testbed for a Satellite Computer Network with Laser Cross Links." Research Report Strategic US Army Space and Strategic Defense Command DASG60-89-C-0119 (Oct.).

Crowell, F. and N. Elzenga. 1990. "Transputer Link Communication" In *A Technical Report* SGS-Thomson Microelectronics, Santa Ana California.

Hoare, C.A.R. 1985. *Communicating Sequential Processes.*

Khan, S.A. and C. Ward. 1991. "Design Considerations for a High Speed Packet Routing Switch Utilizing Transputers." In *Proceedings of the 1991 International Simulation Technology Conference* (Orlando, FL, Oct. 21-23) Society for Computer Simulation, San Diego, California, 418-428.

Paul, D. and R. Marshalek. 1989. *Forward Error correction and its impact on High-data-rate, Free-space Laser Communication System Design* COMSAT Laboratories, Clarkburg, Maryland.

Transtech Devices Limited. 1990. *Genesys Command Reference Manual.*

Ward, C. and S.A. Khan. 1991. "Design of a High Speed Packet Routing Switch Utilizing Transputers." In *Applications of Transputers* 3 no. 2 (Aug.): 469-474.

Ward, C.; S.A. Khan; Y.C. Chow and R.E. Newman-Wolfe. 1992. "Performance of a Routing Switch Utilizing Transputers." In *Proceedings of the 1992 International Simulation Technology Conference* (Clear Lake, TX, Nov. 4-6) Society for Computer Simulation, San Diego, California, 118-132.

Ward, C. and C.H. Choi. 1991. "The LAMS-DLC Protocol." *Computer Communications Review* 21, no. 4 (Sept.): 249-257.

Appendix A CSP Specifications for Satellite Model

In the following satellite model process specifications the highest level of abstraction used parallels the transputer architecture presented in the paper. Details were omitted to focus on inter-process data communication flows. More specifically, variable assignments, loop controls, and control signal traffic are omitted to provide a clearer understanding of data traffic flows.

```
SATELLITE MODEL SPECIFICATION

SATELLITE_MODEL =

  HOST                          -- Host processes
  ||                            -- in PARALLEL with
  BRIDGE                        -- Bridge processes
  ||                            -- in PARALLEL with
  ROUTERS                       -- Router processes
  ||                            -- in PARALLEL with
  DRIVERS                       -- Driver processes
```

This formalism specifies the object on the left side of the equality, namely SATELLITE_MODEL, to be composed of a host, a bridge, some routers, and some drivers. Abstract processes are identifiable by all capital letters in their name. Underscores are used to join multiple word names. The parallel bars, "||", indicate that the processes are to be executed concurrently. At each new level of abstraction, processes from the previous level are refined to add more detail, as follows:

```
HOST PROCESS SPECIFICATION

HOST =

  MULTIPLEX_HOST_OUTPUTS        -- Transmit signals, data and
                                --   routing packets
  ||                            -- in PARALLEL with
  GENERATE_ROUTING_PACKETS      -- Generate routing table updates
  ||                            -- in PARALLEL with
```

```
  GENERATE_DATA_PACKETS          -- Generate internal data packets
  ||                             -- in PARALLEL with
  RECEIVE_HOST_INPUTS            -- Receiving signals & data packets
```

In the following specifications, the formalism ";" is used to specify sequential execution. Prefix execution is specified with "->", indicating subsequent execution of the following statement(s), if the preceeding event takes place. To smooth the effects of eliminating control information, simple recursion is used and is annotated by the formalism "u.". The (condition)*(statements) formalism is used to represent looping . The "true branch <| condition |> false branch " formalism represents conditional branching, noting that the true branch precedes the condition. "?" and "!" are used to indicate channel input and output, respectively. And, "|" is used to denote choice by channel selection.

```
u.MULTIPLEX_HOST_OUTPUTS =
 (internal.channel.1 ?           -- Receive/Send route update
    route.update;router.id;router.destination;new.rte ->
  channel.to.bridge !
    route.update;router.id;router.destination;new.rte
  |                              -- in ALTERNATE with
                                 -- Receive/Send data packet
  internal.channel.2 ? external.data;destination;length::data ->
  channel.to.bridge ! external.data;destination;length::data);
  MULTIPLEX_HOST_OUTPUTS

u.GENERATE_ROUTING_PACKETS =
  timer := timer PLUS delay1;    -- Add route update interpacket delay
  timer.channel ? AFTER timer;   -- Delay until next arrival time
  internal.channel.1 !           -- Send routing table update
    route.update;brdg;destination;new.route;
  GENERATE_ROUTING_PACKETS
```

```
GENERATE_DATA_PACKETS =
  (more.data.to.send) * (          -- While more host data to send
    timer := timer PLUS delay2;    --   Add data interpacket delay time
                                   --   Delay until next arrival time
    timer.channel ? AFTER timer;
    internal.channel.2 !           --   Send external data packet
      external.data;destination;length::data);
  SKIP


u.RECEIVE_HOST_INPUTS =            -- Receive external data packets
  channel.from.bridge ? external.data;destination;lenght::data ->
  SAVE_THROUGHPUT_DATA;            -- Store interpacket timing data
  RECEIVE_HOST_INPUTS
```

BRIDGE PROCESS SPECIFICATION

```
BRIDGE =
  MULTIPLEX_TRAFFIC_TO_HOST        -- Send Host control/data traffic
  ||                               -- in PARALLEL with
  DEMULTIPLEX_TRAFFIC_FROM_HOST    -- Receive Host control/data traffic
  ||                               -- in PARALLEL with
  MULTIPLEX_ROUTER0_TRAFFIC        -- Send Router0 control/data traffic
  ||                               -- in PARALLEL with
  MULTIPLEX_ROUTER2_TRAFFIC        -- Send Router2 control/data traffic


u.MULTIPLEX_TRAFFIC_TO_HOST =      -- Send Host control/data traffic
 (internal.channel.1 ? internal.data;destination;length::data ->
  channel.to.host ! external.data;destination;length::data
```

```
  |
  internal.channel.2 ? internal.data;destination;length::data ->
  channel.to.host ! external.data;destination;length::data);
  MULTIPLEX_HOST_TRAFFIC

u.DEMULTIPLEX_TRAFFIC_FROM_HOST  -- Receive Host control/data traffic
  channel.from.host ? external.data;destination;length::data ->
 (internal.channel.3 ! internal.data;destination;length::data
  <| dest = RTR0 OR dest = RTR1 |>
  internal.channel.4 ! internal.data;destination;length::data);
  DEMULTIPLEX_TRAFFIC_FROM_HOST

u.MULTIPLEX_ROUTER0_TRAFFIC =    -- Send Router0 control/data traffic
 (channel.from.router2 ? external.data;destination;length::data ->
  channel.to.router0 ! external.data;destination;length::data
  |
  channel.from.router2 ? internal.data;destination;length::data ->
  internal.channel.1 ! internal.data;destination;length::data
  |
  internal.channel.3 ? internal.data;destination;length::data ->
  channel.to.router0 ! internal.data;destination;length::data);
  MULTIPLEX_ROUTER0_TRAFFIC

u.MULTIPLEX_ROUTER2_TRAFFIC =    -- Send Router2 control/data traffic
 (channel.from.router0 ? external.data;destination;length::data ->
  channel.to.router2 ! external.data;destination;length::data
  |
  channel.from.router0 ? internal.data;destination;length::data ->
  internal.channel.2 ! internal.data;destination;length::data
```

```
  |
  internal.channel.4 ? internal.data;destination;length::data ->
  channel.to.router2 ! internal.data;destination;length::data);
  MULTIPLEX_ROUTER2_TRAFFIC
```

ROUTER PROCESS SPECIFICATION

```
ROUTERS = || (for i = 0 to 3) ROUTER(i)
```

The "(FOR variable = range)" formalism declares four processes, namely ROUTER(0) through ROUTER(3), to be executed in parallel.

```
ROUTER(i) =
  MULTIPLEX_TRAFFIC_TO_DRIVER(i) -- Send Driver control/data traffic
  ||                             -- in PARALLEL with
  ROUTE_NETWORK_TRAFFIC          -- Route router thru-traffic
  ||                             -- in PARALLEL with
  DEMULTIPLEX_HOST_TRAFFIC       -- Receive host traffic
  ||                             -- in PARALLEL with
  DEMULTIPLEX_ROUTE_A_TRAFFIC    -- Receive routeA traffic

                                 -- Multiplex traffic to driver

u.MULTIPLEX_TRAFFIC_TO_DRIVER(i) =
 (internal.channel.1 ? external.data;destination;length::data ->
  channel.to.driver(i) ! external.data;destination;length::data
  |
  internal.channel.2 ? external.data;destination;length::data ->
  channel.to.driver(i) ! external.data;destination;length::data
```

```
  |
  internal.channel.5 ? external.data;destination;length::data ->
  channel.to.driver(i) ! external.data;destination;length::data);
  MULTIPLEX_TRAFFIC_TO_DRIVER(i)

u.ROUTE_NETWORK_TRAFFIC =          -- Route traffic to other routers
  (internal.channel.3 ? internal.route;destination;route ->
  (UPDATE_ROUTING_TABLE;
   channel.to.routeA ! internal.route;destination;route
   <| i = 1 OR 3 |>
   SKIP)
   |
   internal.channel.3 ? internal.data;destination;length::data ->
  (channel.to.routeA ! external.data;destination;length::data
   <| table(destination) = RTR(i).RTA |>
  (internal.channel.5 ! external.data;destination;length::data
   <| table(destination) = RTR(i).DRV |>
   SKIP))
   |
   internal.channel.4 ? internal.data;destination;length::data ->
   channel.to.host ! internal.data;destination;length::data
   |
   channel.from.driver(i) ? external.data;destination;length::data ->
 ((channel.to.bridge ! internal.data;destination;length::data
   <| destination = HOST |>
   channel.to.bridge ! external.data;deswtination;length::data)
   <| table(destination) = RTR(i).HST |>
  (channel.to.routeA ! external.data;destination;length::data
   <| table(destination) = RTR(i).RTA |>
```

```
   (channel.to.routeB ! external.data;destination;length::data
    <| table(destination) = RTR(i).RTB |>
    SKIP))));
    ROUTE_NETWORK_TRAFFIC

u.DEMULTIPLEX_HOST_TRAFFIC =      -- Receive inputs from host path
   (channel.from.host ? internal.route;destination;route ->
      internal.channel.3 ! internal.route;destination;route
    |
    channel.from.host ? external.data;destination;length::data ->
      internal.channel.1 ! external.data;destination;length::data
    |
    channel.from.host ? internal.data;destination;length::data ->
      internal.channel.3 ! internal.data;destination;length::data);
    DEMULTIPLEX_HOST_TRAFFIC

u.DEMULTIPLEX_ROUTE_A_TRAFFIC =   -- Receive inputs from routeA path
   (channel.from.routeA ? external.data;destination;length::data ->
    internal.channel.2 ! external.data;destination;length::data
    |
    channel.from.routeA ? internal.data;destination;length::data ->
    internal.channel.4 ! internal.data;destination;length::data);
    DEMULTIPLEX_ROUTE_A_TRAFFIC
```

DRIVER PROCESS SPECIFICATIONS

```
DRIVERS = || (for i = 0 to 3) DRIVER(i)
```

```
u.DRIVER(i) =
 (SEND_DRIVER(i)_DATA              -- Send driver control/data traffic
  ||
  RECEIVE_ROUTER(i)_TRAFFIC);      -- Receive driver cntrl/data traffic
  SEND_RESULTS;
  DRIVER(i);

SEND_DRIVER(i)_DATA =
 (more.data.to.send) * (           -- While more host data to send
    timer = timer PLUS delay;      --   Add data interpacket delay time
   .timer.channel ? AFTER timer;-- Delay until next arrival time
    channel.to.router(i) !         --   Send external data packet
      external.data;destination;length::data);
  SKIP

u.RECEIVE_ROUTER(i)_TRAFFIC =    -- Receive driver traffic from router
  channel.from.router(i) ? external.data;destination;length::data ->
  perform.throughput.calculations;
  RECEIVE_ROUTER(i)_TRAFFIC
```

Appendix B Occam Source Code for Satellite Model

```
-- hardware description
VAL K IS 1024:
VAL M IS K * K:

NODE ho, br, r0, r1, r2, r3, d0, d1, d2, d3 :
ARC HOSTLINK, ho.br, d0.r0, r0.r1, r1.d1, d2.r2, r2.r3, r3.d3, br.r2, br.r0, r0.r3, r1.r
3, r2.r1 :
NETWORK
  DO
    SET ho (type, memsize := "T800", 1 * M)
    SET br (type, memsize := "T800", 1 * M)
    SET r0 (type, memsize := "T800", 1 * M)
    SET r1 (type, memsize := "T800", 1 * M)
    SET r2 (type, memsize := "T800", 1 * M)
    SET r3 (type, memsize := "T800", 1 * M)
    SET d0 (type, memsize := "T800", 1 * M)
    SET d1 (type, memsize := "T800", 1 * M)
    SET d2 (type, memsize := "T800", 1 * M)
    SET d3 (type, memsize := "T800", 1 * M)
    CONNECT ho[link][1] TO HOST        WITH HOSTLINK
    CONNECT ho[link][2] TO br[link][1] WITH ho.br
    CONNECT d0[link][2] TO r0[link][1] WITH d0.r0
    CONNECT r0[link][2] TO r1[link][1] WITH r0.r1
    CONNECT r1[link][2] TO d1[link][1] WITH r1.d1
    CONNECT d2[link][2] TO r2[link][1] WITH d2.r2
    CONNECT r2[link][2] TO r3[link][1] WITH r2.r3
    CONNECT r3[link][2] TO d3[link][1] WITH r3.d3
    CONNECT br[link][0] TO r2[link][0] WITH br.r2
    CONNECT br[link][3] TO r0[link][0] WITH br.r0
    CONNECT r0[link][3] TO r3[link][0] WITH r0.r3
    CONNECT r2[link][3] TO r1[link][0] WITH r2.r1
    CONNECT r1[link][3] TO r3[link][3] WITH r1.r3
:

-- software description
#INCLUDE "hostio.inc"
#INCLUDE "protocol.inc"
#USE "host.c8h"
#USE "bridge.c8h"
#USE "router0.c8h"
#USE "router1.c8h"
#USE "router2.c8h"
#USE "router3.c8h"
#USE "driver0.c8h"
#USE "driver1.c8h"
#USE "driver2.c8h"
#USE "driver3.c8h"

CONFIG
  CHAN OF SP fs,ts:
  [24]CHAN OF MESSAGE ln:

  PLACE fs,ts ON HOSTLINK :

  PAR
    PROCESSOR ho
      HOST(fs, ts, ln[0], ln[1])
    PROCESSOR br
      BRIDGE(ln[14], ln[1], ln[16], ln[15], ln[0], ln[17])
    PROCESSOR r0
      ROUTER0(ln[17], ln[3], ln[4], ln[18], ln[16], ln[2], ln[5], ln[19])
    PROCESSOR r1
      ROUTER1(ln[5], ln[6], ln[21], ln[22], ln[4], ln[7], ln[20], ln[23])
    PROCESSOR r2
      ROUTER2(ln[15], ln[9], ln[10], ln[20], ln[14], ln[8], ln[11], ln[21])
    PROCESSOR r3
      ROUTER3(ln[11], ln[12], ln[19], ln[23], ln[10], ln[13], ln[18], ln[22])
    PROCESSOR d0
      DRIVER0(ln[2], ln[3])
    PROCESSOR d1
      DRIVER1(ln[7], ln[6])
    PROCESSOR d2
      DRIVER2(ln[8], ln[9])
    PROCESSOR d3
      DRIVER3(ln[13], ln[12])

:
```

config


```
r 0
r 1
r 2
c 0.2.0 2.2.0
c 0.2.3 1.2.0
c 1.2.3 2.3.0
c 1.3.0 2.2.3
c 1.3.3 2.3.3
s 0 0
s 1 5
s 2 5
q
```

```
PROTOCOL MESSAGE
  CASE
    ext.dat; INT; INT::[]BYTE                          -- dest, message
    int.dat; INT; INT::[]BYTE                          -- dest, message
    int.rte; INT; INT                                  -- dest, route
    report
    results; INT; INT; INT; INT; INT; REAL64; REAL64; REAL64; INT
                                                       -- ID,SNO,DSTR,DLY,T0,E,E2,E3,PR
    stage.completion
    synchronize
    test.completion
    terminate
:
```

```
-- TEST SETUP CONFIGURATION
VAL NO IS 0:
VAL YES IS 1:
VAL NONE IS 0:
VAL MINSET IS 1:
VAL OVERHEAD IS 1:
VAL SECOND IS 15625:
VAL MLEN IS 997:
VAL TOT.DESTS IS 5:
VAL HOST.RNO IS 2:

-- DESTINATION DECLARATIONS
VAL HOST IS 0:
VAL BRDG IS 0:
VAL RTR0 IS 1:
VAL DRV0 IS 1:
VAL RTR1 IS 2:
VAL DRV1 IS 2:
VAL RTR2 IS 3:
VAL DRV2 IS 3:
VAL RTR3 IS 4:
VAL DRV3 IS 4:
VAL RNDM IS 5:

-- HOST ROUTING UPDATES
VAL HRTE.SND IS NO:
VAL HRTE.DST IS NONE:
VAL HRTE.DLY IS NONE:
VAL HRTE.RTE IS NONE:

VAL CDST IS NONE:          -- common destination ID
VAL CSNO IS NONE:          -- common no# of packets to send
VAL CPAT IS NONE:          -- common packet arrival time

-- HOST DATA PACKETS
VAL HOST.DST IS NONE:
VAL HOST.SNO IS NONE:
VAL HOST.DLY IS NONE:

-- DRIVER0
VAL DRV0.DST IS NONE:
VAL DRV0.SNO IS NONE:
VAL DRV0.DLY IS NONE:

-- DRIVER1
VAL DRV1.DST IS HOST:
VAL DRV1.SNO IS 5000:
VAL DRV1.DLY IS 0:

-- DRIVER2
VAL DRV2.DST IS NONE:
VAL DRV2.SNO IS NONE:
VAL DRV2.DLY IS NONE:

-- DRIVER3
VAL DRV3.DST IS NONE:
VAL DRV3.SNO IS NONE:
VAL DRV3.DLY IS NONE:

-- BRIDGE
VAL BRDG.HST IS 1:                    -- Physical channel
VAL BRDG.RTA IS 0:
VAL BRDG.RTB IS 3:
VAL BRDG.TB0 IS 1:                    -- Route to dest 0
VAL BRDG.TB1 IS 3:
VAL BRDG.TB2 IS 3:
VAL BRDG.TB3 IS 0:
VAL BRDG.TB4 IS 0:
VAL [TOT.DESTS]INT BRDG.TAB IS [BRDG.TB0,BRDG.TB1,BRDG.TB2,BRDG.TB3,BRDG.TB4]:

-- ROUTER0
VAL RTR0.HST IS 0:                    -- Physical channel
VAL RTR0.DRV IS 1:                    -- Physical channel
VAL RTR0.RTA IS 2:
VAL RTR0.RTB IS 3:
VAL RTR0.TB0 IS 0:                    -- Route to dest 0
VAL RTR0.TB1 IS 1:
VAL RTR0.TB2 IS 2:
VAL RTR0.TB3 IS 0:
VAL RTR0.TB4 IS 3:
VAL [TOT.DESTS]INT RTR0.TAB IS [RTR0.TB0,RTR0.TB1,RTR0.TB2,RTR0.TB3,RTR0.TB4]:

-- ROUTER1
VAL RTR1.HST IS 1:
VAL RTR1.DRV IS 2:
VAL RTR1.RTA IS 0:                    -- Other member of routerpair
VAL RTR1.RTB IS 3:
VAL RTR1.TB0 IS 1:
VAL RTR1.TB1 IS 1:
VAL RTR1.TB2 IS 2:
VAL RTR1.TB3 IS 0:
VAL RTR1.TB4 IS 3:
VAL [TOT.DESTS]INT RTR1.TAB IS [RTR1.TB0,RTR1.TB1,RTR1.TB2,RTR1.TB3,RTR1.TB4]:

-- ROUTER2
VAL RTR2.HST IS 0:
VAL RTR2.DRV IS 1:
VAL RTR2.RTA IS 2:
VAL RTR2.RTB IS 3:
VAL RTR2.TB0 IS 0:
VAL RTR2.TB1 IS 0:
VAL RTR2.TB2 IS 3:
VAL RTR2.TB3 IS 1:
VAL RTR2.TB4 IS 2:
VAL [TOT.DESTS]INT RTR2.TAB IS [RTR2.TB0,RTR2.TB1,RTR2.TB2,RTR2.TB3,RTR2.TB4]:

-- ROUTER3
VAL RTR3.HST IS 1:
VAL RTR3.DRV IS 2:
VAL RTR3.RTA IS 0:
VAL RTR3.RTB IS 3:
VAL RTR3.TB0 IS 1:
VAL RTR3.TB1 IS 0:
VAL RTR3.TB2 IS 3:
VAL RTR3.TB3 IS 1:
VAL RTR3.TB4 IS 2:
VAL [TOT.DESTS]INT RTR3.TAB IS [RTR3.TB0,RTR3.TB1,RTR3.TB2,RTR3.TB3,RTR3.TB4]:
```

```
#INCLUDE "hostio.inc"
#INCLUDE "protocol.inc"      -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"    -- test configuration parameters
#INCLUDE "HOST_DST.inc"      -- host packet destination array
#INCLUDE "HOST_DLY.inc"      -- host interpacket arrival delay times array
#INCLUDE "HOST_LEN.inc"      -- host data packet message length array
#USE "hostio.lib"
#USE "dblmath.lib"

--*************************************************************************
--*  This process drives the HOST communications link.  It sends routing  *
--*  table updates and data packets at predetermined time intervals (set in *
--*  test_setup.inc), receives data and signals from the drivers, prints out *
--*  stage results, and sends signals to control overall execution.       *
--*************************************************************************

PROC HOST(CHAN OF SP fs, ts, CHAN OF MESSAGE ho.ot, ho.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7:

  INT time1, time2, delay1, delay2, cnt, tcnt, id, sno, dst, dly, t0, pr:
  REAL64 e, e2, e3, v, ar, tp, val1, val2:
  [MLEN]BYTE msg:
  BOOL going:
  SEQ
    so.write.string.nl(fs,ts,"                                        _____")
    so.write.string.nl(fs,ts,"                      | <<<<  DESTINATION ID TABLE  >>>>> |"
)
    so.write.string.nl(fs,ts,"                      |-----------------------------------|"
)
    so.write.string.nl(fs,ts,"                      | #0  =  HOST     |  #3  =  DRIVER2 |"
)
    so.write.string.nl(fs,ts,"                      | #1  =  DRIVER0  |  #4  =  DRIVER3 |"
)
    so.write.string.nl(fs,ts,"                      | #2  =  DRIVER1  |  #5  =  RANDOM  |"
)
    so.write.string.nl(fs,ts,"                      |_________________|_________________|"
)
    so.write.nl(fs,ts)
    so.write.string.nl(fs,ts,"<<<<<<<<<<<<<<<<<<<<<<<<  B E G I N   O F   T E S T  >>>>
>>>>>>>>>>>>>>>>>>>>>>")
    so.write.nl(fs,ts)
    delay1 := HRTE.DLY                  -- set data interpacket delay
    delay2 := HOST.DLY                  -- set routing update delay
    going := TRUE
    WHILE going
      SEQ
        PAR

          -- Generate internal data packets
          TIMER clock:
          INT cnt, cnt2, timer:
          SEQ
            cnt, cnt2 := 0, 0
            int.com1 ? CASE             -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE  ((HOST.SNO > cnt) AND (delay2 >= 0))
              SEQ
                IF
                  HOST.DST = RNDM       -- for randomized data test runs
                    SEQ
                      WHILE (dest[cnt2] = HOST)
                        IF
                          cnt2 = (HOST.SNO - 1)
                            cnt2 := 0
                          TRUE
                            cnt2 := cnt2 + 1
                      timer := timer PLUS (i[cnt2] / 1000)
                      clock ? AFTER timer
                                        -- send data packet to MUX
                      int.com5 ! ext.dat;dest[cnt2];n[cnt2]::msg
                      IF
                        cnt2 = (HOST.SNO - 1)
                          cnt2 := 0
                        TRUE
                          cnt2 := cnt2 + 1
                  TRUE                  -- for constant data test runs
                    SEQ
                      timer := timer PLUS delay2
                      clock ? AFTER timer
                                        -- send data packet to MUX
                      int.com5 ! ext.dat;HOST.DST;MLEN::msg
                  cnt := cnt + 1
            IF                          -- notify receiver - sender is done!
              ((HOST.DST <> RNDM) AND
               (((HRTE.SND = YES)  AND (delay1 > 0)) OR
                ((HOST.SNO > 0)    AND (delay2 > 0))))
                int.com7 ! stage.completion
              TRUE
                int.com7 ! test.completion

          -- Generate routing update packets
          TIMER clock:
          INT timer:
          BOOL going:
          SEQ
            IF
              HRTE.SND = YES
                going := TRUE
              TRUE                      -- if no routing updates, notify MUX
                SEQ
                  int.com6 ! synchronize
                  going := FALSE
            int.com2 ? CASE             -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE (going AND (delay1 >= 0))
              SEQ
                timer := timer PLUS delay1
                clock ? AFTER timer
                                        -- send routing update to MUX
                int.com6 ! int.rte;HRTE.DST;HRTE.RTE
                int.com3 ? CASE
                  synchronize
                    SKIP
                  terminate
                    going := FALSE

          --  Multiplex Host output channel traffic
          INT dest, route, len:
          [MLEN]BYTE msg:
          BOOL going, updating:
          SEQ
            going, updating := TRUE, TRUE
            WHILE going
              ALT                       -- Multiplex between
```

```
      int.com4 ? CASE              --    messages from receiver
        synchronize
          ho.ot ! synchronize
        terminate
          PAR
            IF
              updating
                int.com6 ? CASE
                  int.rte;dest;route
                    int.com3 ! terminate
                  synchronize
                    SKIP
              TRUE
                SKIP
            going := FALSE

      int.com5 ? CASE              --  with data packets
        ext.dat;dest;len::msg
          ho.ot ! ext.dat;dest;len::msg

      int.com6 ? CASE              --  and routing updates
        int.rte;dest;route
          PAR
            ho.ot ! int.rte;dest;route
            int.com3 ! synchronize
        synchronize
          updating := FALSE

--  Receive Host input traffic
TIMER clock:
INT cnt, dest, len, time.last, time.present:
SEQ
  cnt, tcnt, t0, pr := 0, 0, 0, 0
  e, e2, e3 := 0.0(REAL64), 0.0(REAL64), 0.0(REAL64)
  PAR
    int.com4 ! synchronize     --  Resynchronize drivers
    int.com2 ! synchronize     --  Resynchronize routing generator
    int.com1 ! synchronize     --  Resynchronize data generator
  clock ? time1                --  Read starting clock time
  time.last := time1
  WHILE  cnt < HOST.RNO
    ALT                        --  Multiplex between
      ho.in ? CASE             --    host input traffic
        ext.dat;dest;len::msg
          SEQ                  --    & compute/save timing values
            clock ? time.present
            IF
              t0 = 0
                t0 := time.present - time.last
              TRUE
                SEQ
                  e := e + (REAL64 TRUNC (time.present - time.last))
                  e2 := e2 + (e * e)
                  e3 := e3 + (e * e2)
            pr := pr + 1
            time.last := time.present
        stage.completion
          SEQ
            cnt := cnt + 1
        test.completion
          SEQ
            cnt := cnt + 1
            tcnt := tcnt + 1
      int.com7 ? CASE          --  and host completion signal
        stage.completion
          SEQ
            cnt := cnt + 1
        test.completion
          SEQ
            cnt := cnt + 1
            tcnt := tcnt + 1
    -- wrap up general host processes
    clock ? time2              --    Read stopping clock time
    int.com4 ! terminate

  -- Report Host performance
  so.write.string.nl(fs,ts,"------------------------------    HOST    ---------
-----------------------")
  IF
    HOST.SNO > 0
      SEQ
        so.write.string(fs,ts,"Transmitted ")
        so.write.int(fs,ts,HOST.SNO,0)
        so.write.string(fs,ts," packets to DESTINATION ID #")
        so.write.int(fs,ts,HOST.DST,0)
        so.write.nl(fs,ts)
        ar := (REAL64 TRUNC delay2) / (REAL64 TRUNC SECOND)
        so.write.string(fs,ts,"Interpacket Arrival Rate      =  ")
        so.write.real64(fs,ts,ar,0,8)
        so.write.string.nl(fs,ts," seconds")
    TRUE
      so.write.string.nl(fs,ts,"< No Data Packets Transmitted >")
  IF
    HRTE.SND = YES
      SEQ
        IF
          delay1 > 0
            ar := (REAL64 TRUNC delay1) / (REAL64 TRUNC SECOND)
          TRUE
            ar := 0.0(REAL64)
        so.write.string(fs,ts,"Routing update arrival frequency  =  ")
        so.write.real64(fs,ts,ar,0,8)
        so.write.string.nl(fs,ts," seconds")
    TRUE
      so.write.string.nl(fs,ts,"< No Routing Updates Performed >")
  so.write.nl(fs,ts)
  IF
    pr > 0
      SEQ
        so.write.string(fs,ts,"  Total Packets Received  =  ")
        so.write.int(fs,ts,pr,0)
        so.write.nl(fs,ts)
        IF
          pr > 1
            SEQ
              e := e / (REAL64 TRUNC SECOND)
              so.write.string(fs,ts,"  Total Time Expended     =  ")
              so.write.real64(fs,ts,e,0,8)
              so.write.string.nl(fs,ts," s")
              e := e / (REAL64 TRUNC (pr-1))
              so.write.string(fs,ts,"           Mean
 =  ")
              so.write.real64(fs,ts,e,0,8)
              so.write.string.nl(fs,ts," s")
              e2 := e2 / (REAL64 TRUNC ((REAL64 TRUNC ((pr-1) * (pr-1))) * (REAL64
TRUNC (SECOND * SECOND))))
              so.write.string(fs,ts,"           2nd moment      =  ")
```

```
                so.write.real64(fs,ts,e2,0,8)
                so.write.nl(fs,ts)
                e3 := e3 / (REAL64 TRUNC (((REAL64 TRUNC ((pr-1) * (pr-1))) * (REAL6
4 TRUNC (pr-1))) * ((REAL64 TRUNC (SECOND * SECOND)) * (REAL64  TRUNC SECOND))))
                so.write.string(fs,ts,"          3rd moment      = ")
                so.write.real64(fs,ts,e3,0,8)
                so.write.nl(fs,ts)
                v := DSQRT(DABS(e2 - (e*e)))
                so.write.string(fs,ts,"          Std Deviation   = ")
                so.write.real64(fs,ts,v,0,8)
                so.write.nl(fs,ts)
                tp := (REAL64 TRUNC t0) / (REAL64 TRUNC SECOND)
                so.write.string(fs,ts,"  End-End Delay (0th Pkt) = ")
                so.write.real64(fs,ts,tp,0,8)
                so.write.string.nl(fs,ts," s")
                tp := 8016.0(REAL64) / e
                so.write.string(fs,ts,"  Total Throughput Rate
  = ")
                so.write.real64(fs,ts,tp,0,8)
                so.write.string.nl(fs,ts," bits/s")
              TRUE
                so.write.string.nl(fs,ts,"< Insufficient Throughput Data >")
            TRUE
              so.write.string.nl(fs,ts,"< No Data Packets Recei      ")
          so.write.nl(fs,ts)

          -- Report Driver performance
          ho.ot ! report                    --  Signal drivers to report results
          cnt := 1
          WHILE  cnt < HOST.RNO
            ho.in ? CASE
              results;id;sno;dst;dly;t0;e;e2;e3;pr
                SEQ
                  so.write.string(fs,ts,"------------------------------   DRIVER")
                  so.write.int(fs,ts,id-1,0)
                  so.write.string.nl(fs,ts,"   --------------------------------")
                  IF
                    sno > 0
                      SEQ
                        so.write.string(fs,ts,"Transmitted ")
                        so.write.int(fs,ts,sno,0)
                        so.write.string(fs,ts," packets to DESTINATION ID #")
                        so.write.int(fs,ts,dst,0)
                        so.write.nl(fs,ts)
                        ar := (REAL64 TRUNC dly) / (REAL64 TRUNC SECOND)
                        so.write.string(fs,ts,"Interpacket Arrival Rate       = ")
                        so.write.real64(fs,ts,ar,0,8)
                        so.write.string.nl(fs,ts," seconds")
                    TRUE
                      so.write.string.nl(fs,ts,"< No Data Packets Transmitted >")
                  so.write.nl(fs,ts)
                  IF
                    pr > 0
                      SEQ
                        so.write.string(fs,ts,"  Total Packets Received  =  ")
                        so.write.int(fs,ts,pr,0)
                        so.write.nl(fs,ts)
                        IF
                          pr > 1
                            SEQ
                              e := e / (REAL64 TRUNC SECOND)
                              so.write.string(fs,ts,"  Total Time Expended     = ")
                              so.write.real64(fs,ts,e,0,8)
                              so.write.string.nl(fs,ts," s")
                              e := e / (REAL64 TRUNC (pr-1))
                              so.write.string(fs,ts,"          Mean
        = ")
                              so.write.real64(fs,ts,e,0,8)
                              so.write.string.nl(fs,ts," s")
                              e2 := e2 / (REAL64 TRUNC ((REAL64 TRUNC ((pr-1) * (pr-1))) *
(REAL64 TRUNC (SECOND * SECOND))))
                              so.write.string(fs,ts,"          2nd moment      = ")
                              so.write.real64(fs,ts,e2,0,8)
                              so.write.nl(fs,ts)
                              e3 := e3 / (REAL64 TRUNC (((REAL64 TRUNC ((pr-1) * (pr-1))) *
 (REAL64 TRUNC (pr-1))) * ((REAL64 TRUNC (SECOND * SECOND)) * (REAL64  TRUNC SECOND))))
                              so.write.string(fs,ts,"          3rd moment      = ")
                              so.write.real64(fs,ts,e3,0,8)
                              so.write.nl(fs,ts)
                              v := DSQRT(DABS(e2 - (e*e)))
                              so.write.string(fs,ts,"          Std Deviation   = ")
                              so.write.real64(fs,ts,v,0,8)
                              so.write.nl(fs,ts)
                              tp := (REAL64 TRUNC t0) / (REAL64 TRUNC SECOND)
                              so.write.string(fs,ts,"  End-End Delay (0th Pkt) = ")
                              so.write.real64(fs,ts,tp,0,8)
                              so.write.string.nl(fs,ts," s")
                              tp := 8016.0(REAL64) / e
                              so.write.string(fs,ts,"  Total Throughput Rate
        = ")
                              so.write.real64(fs,ts,tp,0,8)
                              so.write.string.nl(fs,ts," bits/s")
                          TRUE
                            so.write.string.nl(fs,ts,"< Insufficient Throughput Data >")
                    TRUE
                      so.write.string.nl(fs,ts,"< No Data Packets Received >")
                  so.write.nl(fs,ts)
                  cnt := cnt + 1

          -- Decrease data delay (if greater than 0)
          IF
            ((HOST.SNO > 0) AND (delay2 > 0))
              delay2 := delay2 - 10
            TRUE
              -- or Decrease route update delay (if greater than 0)
              IF
                ((HRTE.SND = YES) AND (delay1 > 0))
                  delay1 := delay1 - 10
                TRUE
                  SKIP

          --  Print out total stage comparison results
          so.write.string.nl(fs,ts,"-------------------------  Overall System Performance  -
-------------------------")
          val1 := (REAL64 TRUNC ((((HOST.SNO+DRV0.SNO)+DRV1.SNO)+DRV2.SNO)+ DRV3.SNO)) * 80
16.0(REAL64)
          so.write.string(fs,ts,"Total data transmitted  = ")
          so.write.real64(fs,ts,val1,0,8)
          so.write.string.nl(fs,ts," bits")
          so.write.string(fs,ts,"Starting clock time     = ")
          so.write.int(fs,ts,time1,0)
          so.write.string.nl(fs,ts," ticks")
          so.write.string(fs,ts,"Stopping clock time     = ")
          so.write.int(fs,ts,time2,0)
          so.write.string.nl(fs,ts," ticks")
          val2 := (REAL64 TRUNC((time2 - OVERHEAD) - time1)) / (REAL64 TRUNC SECOND)
          so.write.string(fs,ts,"Total time elapsed      = ")
          so.write.real64(fs,ts,val2,0,8)
```

```
        so.write.string.nl(fs,ts,"s")
        IF
          val2 > 0.0(REAL64)
            val1 := val1 / val2
          TRUE
            val1 := 0.0(REAL64)
        so.write.string(fs,ts,"System Throughput Rate                    = ")
        so.write.real64(fs,ts,val1,0,8)
        so.write.string.nl(fs,ts," bits/s")
        so.write.nl(fs,ts)
        IF
          (HOST.DST <> RNDM) AND (tcnt < HOST.RNO)
            SEQ
              so.write.string.nl(fs,ts,"<<<<<<<<<<<<<<<<<<<<<<<<  E N D    O F    S T A
G E  >>>>>>>>>>>>>>>>>>>>>>>>>")
              so.write.nl(fs,ts)
          TRUE
            SEQ
              -- Send termination signal
              so.write.string.nl(fs,ts,"<<<<<<<<<<<<<<<<<<<<<<<<  E N D    O F    T E S
 T  >>>>>>>>>>>>>>>>>>>>>>>>>")
              so.write.nl(fs,ts)
              going := FALSE
              ho.ot ! terminate
    so.exit(fs,ts,sps.success)          -- EXIT PROGRAM
:
```

```
#INCLUDE "protocol.inc"    -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"  -- test configuration parameters

--*************************************************************************
--*  This process bridges HOST control, data, and routing update traffic  *
--*  with the message traffic on the router network between ROUTER0 and    *
--*  ROUTER2.                                                              *
--*************************************************************************

PROC BRIDGE(CHAN OF MESSAGE r1.ot,ho.ot,r2.ot,r1.in,ho.in,r2.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7,
int.com8:

  PAR

    -- Multiplex Host Outputs
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT                                 -- Multiplex between

          int.com1 ? CASE                   --   communications from Router0
            int.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

          int.com2 ? CASE                   --   and communications from Router2
            int.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

    -- Demultiplex Host Inputs
    [TOT.DESTS]INT table:
    [MLEN]BYTE msg:
    INT dest, route, len:
    BOOL going:
    SEQ
      table := BRDG.TAB
      going := TRUE
      WHILE going

        -- Handle messages arriving on channel 1 (from HOST)
        ho.in ? CASE
          int.rte;dest;route                -- Internal routing updates
            PAR
              table[dest] := BRDG.TAB[route]
              int.com3 ! int.rte;dest;route
              int.com6 ! int.rte;dest;route
          ext.dat;dest;len::msg             -- Internal data messages
            IF
              (table[dest] = BRDG.RTB)
                int.com3 ! int.dat;dest;len::msg
              (table[dest] = BRDG.RTA)
                int.com6 ! int.dat;dest;len::msg
              TRUE
                SKIP
          report                            -- Report signals passed
            PAR
              int.com3 ! report             --   for Router0 and Router1
              int.com6 ! report             --   for Router2 and Router3
          synchronize                       -- Synchronization signals passed
            PAR
              int.com3 ! synchronize        --   for Router0 and Router1
              int.com6 ! synchronize        --   for Router2 and Router3
              table := BRDG.TAB
          terminate                         -- Termination signals passed
            PAR
              going := FALSE
              int.com3 ! terminate          --   for Router0 and Router1
              int.com6 ! terminate          --   for Router2 and Router3

    -- Multiplex Router Outputs (to Router0)
    [MLEN]BYTE msg:
    INT dest, route, len:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT

          -- Handle messages arriving on channel 1 (from HOST)
          int.com3 ? CASE
            int.rte;dest;route              -- Internal routing updates
              r2.ot ! int.rte;dest;route
            int.dat;dest;len::msg           -- Internal data messages
              r2.ot ! int.dat;dest;len::msg
            report                          -- Report signals passed
              r2.ot ! report                --   for Router0 and Router1
            synchronize                     -- Synchronization signals passed
              r2.ot ! synchronize           --   for Router0 and Router1
            terminate                       -- Termination signals passed
              PAR
                going := FALSE
                r2.ot ! terminate           --   for Router0 and Router1
                int.com5 ! terminate        --   for Router0 and Router1

          -- Handle messages arriving on channel 0 (from Router2)
          int.com4 ? CASE
            ext.dat;dest;len::msg           -- External data messages to Router0
              r2.ot ! ext.dat;dest;len::msg

    -- Demultiplex Router Inputs (from Router0)
    [MLEN]BYTE msg:
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64  e, e2, e3:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT
          int.com5 ? CASE                   -- Communications from the Host
            terminate                       --   Termination signals
              PAR
                int.com1 ! terminate
```

```
            going := FALSE

      -- Handle messages arriving on channel 3 (from Router0)
      r2.in ? CASE
        ext.dat;dest;len::msg          -- External data messages to Router2
          int.com7 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg          -- Internal data messages to Host
          int.com1 ! int.dat;dest;len::msg
        stage.completion               -- Stage completion signals to Host
          int.com1 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr -- Results to Host
          int.com1 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion                -- Test completion signals to Host
          int.com1 ! test.completion

-- Multiplex Router Outputs (to Router2)
[MLEN]BYTE msg:
INT dest, route, len:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT
      int.com6 ? CASE                  -- Communications from the Host
        int.rte;dest;route             --   Internal routing updates
          r1.ot ! int.rte;dest;route
        int.dat;dest;len::msg          --   Internal data messages
          r1.ot ! int.dat;dest;len::msg
        report                         --   Report signals
          r1.ot ! report
        synchronize                    --   Synchronization signals
          r1.ot ! synchronize
        terminate                      --   Termination signals
          PAR
            going := FALSE
            r1.ot ! terminate
            int.com8 ! terminate

      -- Handle messages arriving on channel 3 (from Router0)
      int.com7 ? CASE
        ext.dat;dest;len::msg          -- External data messages to Router2
          r1.ot ! ext.dat;dest;len::msg

-- Demultiplex Router Inputs (from Router2)
[MLEN]BYTE msg:
INT dest, id, sno, dst, dly, t0, pr, len:
REAL64 e, e2, e3:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      -- Handle messages arriving on channel 1 (from HOST)
      int.com8 ? CASE
        terminate                      -- Termination signals passed
          going := FALSE

      -- Handle messages arriving on channel 0 (from Router2)
      r1.in ? CASE
        ext.dat;dest;len::msg          -- External data messages to Router0
          int.com4 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg          -- Internal data messages to Host
          int.com2 ! int.dat;dest;len::msg
        stage.completion               -- Stage completion signals to Host
          int.com2 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr  -- Results to Host
          int.com2 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion                -- Test completion signals to Host
          int.com2 ! test.completion
:
```

```
#INCLUDE "protocol.inc"    -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"  -- test configuration parameters

--*************************************************************************
--*  This process bridges Driver control and data traffic onto the router  *
--* network, providing routing services to them along with other message   *
--* traffic passed through the router network.                             *
--*************************************************************************

PROC ROUTER0(CHAN OF MESSAGE ho.ot, dr.ot, r1.ot, r2.ot, ho.in, dr.in, r1.in, r2.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7,
int.com8:

  PAR

    --  Multiplex Driver Outputs
    INT dest, len:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT

          int.com1 ? CASE
            ext.dat;dest;len::msg          -- rerouted host data
              dr.ot ! ext.dat;dest;len::msg
            report
              dr.ot ! report
            synchronize
              dr.ot ! synchronize
            terminate
              PAR
                dr.ot ! terminate
                going := FALSE

          int.com2 ? CASE
            ext.dat;dest;len::msg          -- From Router A
              dr.ot ! ext.dat;dest;len::msg

          r2.in ? CASE                     -- From Router B
            ext.dat;dest;len::msg
              dr.ot ! ext.dat;dest;len::msg

    --  Demultiplex Driver Inputs
    [TOT.DESTS]INT table:
    INT dest, route, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR0.TAB
      going := TRUE
      WHILE going
        ALT

          int.com3 ? CASE                  -- rerouted from the host route
            int.rte;dest;route
              table[dest] := RTR0.TAB[route]
            synchronize
              PAR
                table := RTR0.TAB
            terminate
              PAR
                int.com4 ! terminate
                going := FALSE

          dr.in ? CASE                     -- from the driver
            ext.dat;dest;len::msg
              IF
                table[dest] = RTR0.HST
                  IF
                    dest = HOST
                      int.com4 ! int.dat;dest;len::msg
                    TRUE
                      int.com4 ! ext.dat;dest;len::msg
                table[dest] = RTR0.RTA
                  int.com6 ! ext.dat;dest;len::msg
                table[dest] = RTR0.RTB
                  r2.ot ! ext.dat;dest;len::msg
                TRUE
                  SKIP
            stage.completion
              int.com4 ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              int.com4 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              int.com4 ! test.completion

    --  Multiplex Host Outputs
    [TOT.DESTS]INT table:
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR0.TAB
      going := TRUE
      WHILE going
        ALT

          int.com4 ? CASE                  -- from the driver
            ext.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion
            terminate
              going := FALSE

          int.com5 ? CASE                  -- from Router A
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

    --  Demultiplex Host Inputs
    [TOT.DESTS]INT table:
    INT dest, route, len:
```

```
[MLEN]BYTE msg:
BOOL going:
SEQ
  table := RTR0.TAB
  going := TRUE
  WHILE going

    ho.in ? CASE
      int.rte;dest;route              -- Internal routing updates
        PAR
          table[dest] := RTR0.TAB[route]
          IF
            (RTR0 = 1) OR (RTR0 = 3)
              int.com7 ! int.rte;dest;route
            TRUE
              SKIP
          int.com3 ! int.rte;dest;route
      ext.dat;dest;len::msg           -- External data packets
        int.com1 ! ext.dat;dest;len::msg
      int.dat;dest;len::msg           -- Internal data packets
        IF
          table[dest] = RTR0.RTA
            int.com7 ! ext.dat;dest;len::msg
          table[dest] = RTR0.DRV
            int.com1 ! ext.dat;dest;len::msg
          TRUE
            SKIP
      report                          -- Report signals
        PAR
          int.com1 ! report
          IF
            (RTR0 = 1) OR (RTR0 = 3)
              int.com7 ! report
            TRUE
              SKIP
      synchronize                     -- Synchronize signals
        PAR
          int.com1 ! synchronize
          int.com3 ! synchronize
          IF
            (RTR0 = 1) OR (RTR0 = 3)
              int.com7 ! synchronize
            TRUE
              SKIP
          table := RTR0.TAB
      terminate                       -- Terminate signals
        PAR
          int.com1 ! terminate
          int.com3 ! terminate
          int.com7 ! terminate
          int.com8 ! terminate
          going := FALSE


--  Multiplex Router Outputs (to parter or low-no# channel router)
INT dest, route, len:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com6 ? CASE             -- from the driver
          ext.dat;dest;len::msg
            r1.ot ! ext.dat;dest;len::msg

        int.com7 ? CASE           -- from the host route
          int.rte;dest;route
            r1.ot ! int.rte;dest;route
          ext.dat;dest;len::msg
            r1.ot ! ext.dat;dest;len::msg
          report
            r1.ot ! report
          synchronize
            r1.ot ! synchronize
          terminate
            PAR
              IF
                (RTR0 = 1) OR (RTR0 = 3)
                  r1.ot ! terminate
                TRUE
                  SKIP
              going := FALSE

--  Demultiplex Router Inputs (from partner or low-no# channel router)
INT dest, id, sno, dst, dly, t0, pr, len:
REAL64 e, e2, e3:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com8 ? CASE
        terminate
          going := FALSE

      r1.in ? CASE                -- from Router A
        ext.dat;dest;len::msg
          int.com2 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg
          int.com5 ! int.dat;dest;len::msg
        stage.completion
          int.com5 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr
          int.com5 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion
          int.com5 ! test.completion
:
```

```
#INCLUDE "protocol.inc"    -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"  -- test configuration parameters

--*************************************************************************
--*  This process bridges Driver control and data traffic onto the router  *
--* network, providing routing services to them along with other message   *
--* traffic passed through the router network.                             *
--*************************************************************************

PROC ROUTER1(CHAN OF MESSAGE ho.ot, dr.ot, r1.ot, r2.ot, ho.in, dr.in, r1.in, r2.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7,
int.com8:

  PAR

    --  Multiplex Driver Outputs
    INT dest, len:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT

          int.com1 ? CASE
            ext.dat;dest;len::msg        -- rerouted host data
              dr.ot ! ext.dat;dest;len::msg
            report
              dr.ot ! report
            synchronize
              dr.ot ! synchronize
            terminate
              PAR
                dr.ot ! terminate
                going := FALSE

          int.com2 ? CASE
            ext.dat;dest;len::msg        -- From Router A
              dr.ot ! ext.dat;dest;len::msg

          r2.in ? CASE                   -- From Router B
            ext.dat;dest;len::msg
              dr.ot ! ext.dat;dest;len::msg

    --  Demultiplex Driver Inputs
    [TOT.DESTS]INT table:
    INT dest, route, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR1.TAB
      going := TRUE
      WHILE going
        ALT

          int.com3 ? CASE                -- rerouted from the host route
            int.rte;dest;route
              table[dest] := RTR1.TAB[route]
            synchronize
              PAR
                table := RTR1.TAB
            terminate
              PAR
                int.com4 ! terminate
                going := FALSE

          dr.in ? CASE                   -- from the driver
            ext.dat;dest;len::msg
              IF
                table[dest] = RTR1.HST
                  IF
                    dest = HOST
                      int.com4 ! int.dat;dest;len::msg
                    TRUE
                      int.com4 ! ext.dat;dest;len::msg
                table[dest] = RTR1.RTA
                  int.com6 ! ext.dat;dest;len::msg
                table[dest] = RTR1.RTB
                  r2.ot ! ext.dat;dest;len::msg
                TRUE
                  SKIP
            stage.completion
              int.com4 ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              int.com4 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              int.com4 ! test.completion

    --  Multiplex Host Outputs
    [TOT.DESTS]INT table:
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR1.TAB
      going := TRUE
      WHILE going
        ALT

          int.com4 ? CASE                -- from the driver
            ext.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion
            terminate
              going := FALSE

          int.com5 ? CASE                -- from Router A
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

    --  Demultiplex Host Inputs
    [TOT.DESTS]INT table:
    INT dest, route, len:
```

```
[MLEN]BYTE msg:
BOOL going:
SEQ
  table := RTR1.TAB
  going := TRUE
  WHILE going

    ho.in ? CASE
      int.rte;dest;route                -- Internal routing updates
        PAR
          table[dest] := RTR1.TAB[route]
          IF
            (RTR1 = 1) OR (RTR1 = 3)
              int.com7 ! int.rte;dest;route
            TRUE
              SKIP
          int.com3 ! int.rte;dest;route
      ext.dat;dest;len::msg             -- External data packets
        int.com1 ! ext.dat;dest;len::msg
      int.dat;dest;len::msg             -- Internal data packets
        IF
          table[dest] = RTR1.RTA
            int.com7 ! ext.dat;dest;len::msg
          table[dest] = RTR1.DRV
            int.com1 ! ext.dat;dest;len::msg
          TRUE
            SKIP
      report                            -- Report signals
        PAR
          int.com1 ! report
          IF
            (RTR1 = 1) OR (RTR1 = 3)
              int.com7 ! report
            TRUE
              SKIP
      synchronize                       -- Synchronize signals
        PAR
          int.com1 ! synchronize
          int.com3 ! synchronize
          IF
            (RTR1 = 1) OR (RTR1 = 3)
              int.com7 ! synchronize
            TRUE
              SKIP
          table := RTR1.TAB
      terminate                         -- Terminate signals
        PAR
          int.com1 ! terminate
          int.com3 ! terminate
          int.com7 ! terminate
          int.com8 ! terminate
          going := FALSE


--  Multiplex Router Outputs (to parter or low-no# channel router)
INT dest, route, len:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com6 ? CASE                   -- from the driver
        ext.dat;dest;len::msg
          r1.ot ! ext.dat;dest;len::msg

        int.com7 ? CASE                 -- from the host route
          int.rte;dest;route
            r1.ot ! int.rte;dest;route
          ext.dat;dest;len::msg
            r1.ot ! ext.dat;dest;len::msg
          report
            r1.ot ! report
          synchronize
            r1.ot ! synchronize
          terminate
            PAR
              IF
                (RTR1 = 1) OR (RTR1 = 3)
                  r1.ot ! terminate
                TRUE
                  SKIP
              going := FALSE

--  Demultiplex Router Inputs (from partner or low-no# channel router)
INT dest, id, sno, dst, dly, t0, pr, len:
REAL64 e, e2, e3:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com8 ? CASE
        terminate
          going := FALSE

      r1.in ? CASE                      -- from Router A
        ext.dat;dest;len::msg
          int.com2 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg
          int.com5 ! int.dat;dest;len::msg
        stage.completion
          int.com5 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr
          int.com5 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion
          int.com5 ! test.completion
:
```

```
#INCLUDE "protocol.inc"     -- discriminated protocol for message channels
#INCLUDE "test_setup.inc"   -- test configuration parameters

--*************************************************************************
--*  This process bridges Driver control and data traffic onto the router  *
--* network, providing routing services to them along with other message   *
--* traffic passed through the router network.                             *
--*************************************************************************

PROC ROUTER2(CHAN OF MESSAGE ho.ot, dr.ot, r1.ot, r2.ot, ho.in, dr.in, r1.in, r2.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7,
int.com8:

  PAR

    -- Multiplex Driver Outputs
    INT dest, len:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT

          int.com1 ? CASE
            ext.dat;dest;len::msg          -- rerouted host data
              dr.ot ! ext.dat;dest;len::msg
            report
              dr.ot ! report
            synchronize
              dr.ot ! synchronize
            terminate
              PAR
                dr.ot ! terminate
                going := FALSE

          int.com2 ? CASE
            ext.dat;dest;len::msg          -- From Router A
              dr.ot ! ext.dat;dest;len::msg

          r2.in ? CASE                     -- From Router B
            ext.dat;dest;len::msg
              dr.ot ! ext.dat;dest;len::msg

    -- Demultiplex Driver Inputs
    [TOT.DESTS]INT table:
    INT dest, route, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR2.TAB
      going := TRUE
      WHILE going
        ALT

          int.com3 ? CASE                  -- rerouted from the host route
            int.rte;dest;route
              table[dest] := RTR2.TAB[route]
            synchronize
              PAR
                table := RTR2.TAB
            terminate
              PAR
                int.com4 ! terminate
                going := FALSE

          dr.in ? CASE                     -- from the driver
            ext.dat;dest;len::msg
              IF
                table[dest] = RTR2.HST
                  IF
                    dest = HOST
                      int.com4 ! int.dat;dest;len::msg
                    TRUE
                      int.com4 ! ext.dat;dest;len::msg
                table[dest] = RTR2.RTA
                  int.com6 ! ext.dat;dest;len::msg
                table[dest] = RTR2.RTB
                  r2.ot ! ext.dat;dest;len::msg
                TRUE
                  SKIP
            stage.completion
              int.com4 ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              int.com4 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              int.com4 ! test.completion

    -- Multiplex Host Outputs
    [TOT.DESTS]INT table:
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR2.TAB
      going := TRUE
      WHILE going
        ALT

          int.com4 ? CASE                  -- from the driver
            ext.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion
            terminate
              going := FALSE

          int.com5 ? CASE                  -- from Router A
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

    -- Demultiplex Host Inputs
    [TOT.DESTS]INT table:
    INT dest, route, len:
```

```
[MLEN]BYTE msg:
BOOL going:
SEQ
  table := RTR2.TAB
  going := TRUE
  WHILE going

    ho.in ? CASE
      int.rte;dest;route              -- Internal routing updates
        PAR
          table[dest] := RTR2.TAB[route]
          IF
            (RTR2 = 1) OR (RTR2 = 3)
              int.com7 ! int.rte;dest;route
            TRUE
              SKIP
          int.com3 ! int.rte;dest;route
      ext.dat;dest;len::msg           -- External data packets
        int.com1 ! ext.dat;dest;len::msg
      int.dat;dest;len::msg           -- Internal data packets
        IF
          table[dest] = RTR2.RTA
            int.com7 ! ext.dat;dest;len::msg
          table[dest] = RTR2.DRV
            int.com1 ! ext.dat;dest;len::msg
          TRUE
            SKIP
      report                          -- Report signals
        PAR
          int.com1 ! report
          IF
            (RTR2 = 1) OR (RTR2 = 3)
              int.com7 ! report
            TRUE
              SKIP
      synchronize                     -- Synchronize signals
        PAR
          int.com1 ! synchronize
          int.com3 ! synchronize
          IF
            (RTR2 = 1) OR (RTR2 = 3)
              int.com7 ! synchronize
            TRUE
              SKIP
          table := RTR2.TAB
      terminate                       -- Terminate signals
        PAR
          int.com1 ! terminate
          int.com3 ! terminate
          int.com7 ! terminate
          int.com8 ! terminate
          going := FALSE


--  Multiplex Router Outputs (to parter or low-no# channel router)
INT dest, route, len:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com6 ? CASE             -- from the driver
        ext.dat;dest;len::msg
          r1.ot ! ext.dat;dest;len::msg

      int.com7 ? CASE             -- from the host route
        int.rte;dest;route
          r1.ot ! int.rte;dest;route
        ext.dat;dest;len::msg
          r1.ot ! ext.dat;dest;len::msg
        report
          r1.ot ! report
        synchronize
          r1.ot ! synchronize
        terminate
          PAR
            IF
              (RTR2 = 1) OR (RTR2 = 3)
                r1.ot ! terminate
              TRUE
                SKIP
            going := FALSE

--  Demultiplex Router Inputs (from partner or low-no# channel router)
INT dest, id, sno, dst, dly, t0, pr, len:
REAL64 e, e2, e3:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int com8 ? CASE
        terminate
          going := FALSE

      r1.in ? CASE                -- from Router A
        ext.dat;dest;len::msg
          int.com2 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg
          int.com5 ! int.dat;dest;len::msg
        stage.completion
          int.com5 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr
          int.com5 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion
          int.com5 ! test.completion
:
```

```
#INCLUDE "protocol.inc"    -- discriminated protocol for message channels
#INCLUDE "test_setup.inc"  -- test configuration parameters

--*************************************************************************
--*  This process bridges Driver control and data traffic onto the router  *
--* network, providing routing services to them along with other message   *
--* traffic passed through the router network.                             *
--*************************************************************************

PROC ROUTER3(CHAN OF MESSAGE ho.ot, dr.ot, r1.ot, r2.ot, ho.in, dr.in, r1.in, r2.in)
  CHAN OF MESSAGE int.com1, int.com2, int.com3, int.com4, int.com5, int.com6, int.com7,
int.com8:

  PAR

    -- Multiplex Driver Outputs
    INT dest, len:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      going := TRUE
      WHILE going
        ALT

          int.com1 ? CASE
            ext.dat;dest;len::msg          -- rerouted host data
              dr.ot ! ext.dat;dest;len::msg
            report
              dr.ot ! report
            synchronize
              dr.ot ! synchronize
            terminate
              PAR
                dr.ot ! terminate
                going := FALSE

          int.com2 ? CASE
            ext.dat;dest;len::msg          -- From Router A
              dr.ot ! ext.dat;dest;len::msg

          r2.in ? CASE                     -- From Router B
            ext.dat;dest;len::msg
              dr.ot ! ext.dat;dest;len::msg

    -- Demultiplex Driver Inputs
    [TOT.DESTS]INT table:
    INT dest, route, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR3.TAB
      going := TRUE
      WHILE going
        ALT

          int.com3 ? CASE                  -- rerouted from the host route
            int.rte;dest;route
              table[dest] := RTR3.TAB[route]
            synchronize
              PAR
                table := RTR3.TAB
            terminate
              PAR
                int.com4 ! terminate
                going := FALSE

          dr.in ? CASE                     -- from the driver
            ext.dat;dest;len::msg
              IF
                table[dest] = RTR3.HST
                  IF
                    dest = HOST
                      int.com4 ! int.dat;dest;len::msg
                    TRUE
                      int.com4 ! ext.dat;dest;len::msg
                table[dest] = RTR3.RTA
                  int.com6 ! ext.dat;dest;len::msg
                table[dest] = RTR3.RTB
                  r2.ot ! ext.dat;dest;len::msg
                TRUE
                  SKIP
            stage.completion
              int.com4 ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              int.com4 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              int.com4 ! test.completion

    -- Multiplex Host Outputs
    [TOT.DESTS]INT table:
    INT dest, id, sno, dst, dly, t0, pr, len:
    REAL64 e, e2, e3:
    [MLEN]BYTE msg:
    BOOL going:
    SEQ
      table := RTR3.TAB
      going := TRUE
      WHILE going
        ALT

          int.com4 ? CASE                  -- from the driver
            ext.dat;dest;len::msg
              ho.ot ! ext.dat;dest;len::msg
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion
            terminate
              going := FALSE

          int.com5 ? CASE                  -- from Router A
            int.dat;dest;len::msg
              ho.ot ! int.dat;dest;len::msg
            stage.completion
              ho.ot ! stage.completion
            results;id;sno;dst;dly;t0;e;e2;e3;pr
              ho.ot ! results;id;sno;dst;dly;t0;e;e2;e3;pr
            test.completion
              ho.ot ! test.completion

    -- Demultiplex Host Inputs
    [TOT.DESTS]INT table:
    INT dest, route, len:
```

```
[MLEN]BYTE msg:
BOOL going:
SEQ
  table := RTR3.TAB
  going := TRUE
  WHILE going

    ho.in ? CASE
      int.rte;dest;route              -- Internal routing updates
        PAR
          table[dest] := RTR3.TAB[route]
          IF
            (RTR3 = 1) OR (RTR3 = 3)
              int.com7 ! int.rte;dest;route
            TRUE
              SKIP
          int.com3 ! int.rte;dest;route
      ext.dat;dest;len::msg           -- External data packets
        int.com1 ! ext.dat;dest;len::msg
      int.dat;dest;len::msg           -- Internal data packets
        IF
          table[dest] = RTR3.RTA
            int.com7 ! ext.dat;dest;len::msg
          table[dest] = RTR3.DRV
            int.com1 ! ext.dat;dest;len::msg
          TRUE
            SKIP
      report                          -- Report signals
        PAR
          int.com1 ! report
          IF
            (RTR3 = 1) OR (RTR3 = 3)
              int.com7 ! report
            TRUE
              SKIP
      synchronize                     -- Synchronize signals
        PAR
          int.com1 ! synchronize
          int.com3 ! synchronize
          IF
            (RTR3 = 1) OR (RTR3 = 3)
              int.com7 ! synchronize
            TRUE
              SKIP
          table := RTR3.TAB
      terminate                       -- Terminate signals
        PAR
          int.com1 ! terminate
          int.com3 ! terminate
          int.com7 ! terminate
          int.com8 ! terminate
          going := FALSE


--  Multiplex Router Outputs (to parter or low-no# channel router)
INT dest, route, len:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com6 ? CASE                 -- from the driver
        ext.dat;dest;len::msg
          r1.ot ! ext.dat;dest;len::msg

      int.com7 ? CASE                 -- from the host route
        int.rte;dest;route
          r1.ot ! int.rte;dest;route
        ext.dat;dest;len::msg
          r1.ot ! ext.dat;dest;len::msg
        report
          r1.ot ! report
        synchronize
          r1.ot ! synchronize
        terminate
          PAR
            IF
              (RTR3 = 1) OR (RTR3 = 3)
                r1.ot ! terminate
              TRUE
                SKIP
            going := FALSE

--  Demultiplex Router Inputs (from partner or low-no# channel router)
INT dest, id, sno, dst, dly, t0, pr, len:
REAL64 e, e2, e3:
[MLEN]BYTE msg:
BOOL going:
SEQ
  going := TRUE
  WHILE going
    ALT

      int.com8 ? CASE
        terminate
          going := FALSE

      r1.in ? CASE                    -- from Router A
        ext.dat;dest;len::msg
          int.com2 ! ext.dat;dest;len::msg
        int.dat;dest;len::msg
          int.com5 ! int.dat;dest;len::msg
        stage.completion
          int.com5 ! stage.completion
        results;id;sno;dst;dly;t0;e;e2;e3;pr
          int.com5 ! results;id;sno;dst;dly;t0;e;e2;e3;pr
        test.completion
          int.com5 ! test.completion
:
```

```
#INCLUDE "protocol.inc"     -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"   -- test configuration parameters
#INCLUDE "DRV0_DST.inc"     -- driver packet destination array
#INCLUDE "DRV0_DLY.inc"     -- driver interpacket arrival delay times array
#INCLUDE "DRV0_LEN.inc"     -- driver data packet message length array

--*************************************************************************
--*  This process drives the DRIVER communications link.  It sends data   *
--*  packets at predetermined time intervals (set in test_setup.inc),     *
--*  receives data from the router network, and sends stage results when  *
--*  requested by the HOST.                                               *
--*************************************************************************

PROC DRIVER0(CHAN OF MESSAGE dr.ot, dr.in)
  CHAN OF MESSAGE int.com1:

  INT delay, t0, pr, len:
  REAL64 e, e2, e3:
  BOOL drvr.going:
  SEQ
    delay := DRV0.DLY
    drvr.going := TRUE
    WHILE drvr.going
      SEQ
        PAR

          -- Send next packet when interpacket delay timer expires
          TIMER clock:
          INT cnt, cnt2, timer:
          [MLEN]BYTE msg:
          SEQ
            cnt, cnt2 := 0, 0
            int.com1 ? CASE                 -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE  ((DRV0.SNO > cnt) AND (delay >= 0))
              SEQ
                IF
                  DRV0.DST = RNDM           -- for randomized data test runs
                    SEQ
                      WHILE (dest[cnt2] = DRV0)
                        IF
                          cnt2 = (DRV0.SNO - 1)
                            cnt2 := 0
                          TRUE
                            cnt2 := cnt2 + 1
                      timer := timer PLUS (i[cnt2] / 1000)
                      clock ? AFTER timer
                                          -- send data packet to MUX
                      dr.ot ! ext.dat;dest[cnt2];n[cnt2]::msg
                      IF
                        cnt2 = (DRV0.SNO - 1)
                          cnt2 := 0
                        TRUE
                          cnt2 := cnt2 + 1
                  TRUE                      -- for constant data test runs
                    SEQ
                      timer := timer PLUS delay
                      clock ? AFTER timer
                                          -- send data packet to MUX
                      dr.ot ! ext.dat;DRV0.DST;MLEN::msg
                  cnt := cnt + 1
            IF                              -- notify host - sender is done!
              ((DRV0.DST <> RNDM) AND
               (DRV0.SNO > 0) AND (delay > 0))
                dr.ot ! stage.completion
              TRUE
                dr.ot ! test.completion


          -- Receive a packet and save timing data
          TIMER clock:
          INT dest, time.last, time.present:
          [MLEN]BYTE msg:
          BOOL going:
          SEQ
            t0, pr := 0, 0
            e, e2, e3 := 0.0(REAL64), 0.0(REAL64), 0.0(REAL64)
            going := TRUE
            dr.in ? CASE
              synchronize
                SEQ
                  int.com1 ! synchronize -- Resynchronize data generator
                  clock ? time.last      -- Read starting clock
                  WHILE going
                    SEQ
                      dr.in ? CASE       -- Receive driver input
                        ext.dat;dest;len::msg
                          SEQ            -- Compute/save timing values
                            clock ? time.present
                            IF
                              t0 = 0
                                t0 := time.present - time.last
                              TRUE
                                SEQ
                                  e := e + (REAL64 TRUNC(time.present - time.last))
                                  e2 := e2 + (e * e)
                                  e3 := e3 + (e * e2)
                              pr := pr + 1
                              time.last := time.present
                        report
                          going := FALSE
                  terminate
                    drvr.going := FALSE

          -- Report driver throughput results
          IF
            drvr.going
              SEQ
                dr.ot ! results;DRV0;DRV0.SNO;DRV0.DST;delay;t0;e;e2;e3;pr

                -- Decrease data delay (if greater than 0)
                IF
                  ((DRV0.SNO > 0) AND (delay > 0))
                    delay := delay - 10
                  TRUE
                    SKIP
            TRUE
              SKIP
:
```

```
#INCLUDE "protocol.inc"     -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"   -- test configuration parameters
#INCLUDE "DRV1_DST.inc"     -- driver packet destination array
#INCLUDE "DRV1_DLY.inc"     -- driver interpacket arrival delay times array
#INCLUDE "DRV1_LEN.inc"     -- driver data packet message length array

--*************************************************************************
--*  This process drives the DRIVER communications link.  It sends data   *
--*  packets at predetermined time intervals (set in test_setup.inc),     *
--*  receives data from the router network, and sends stage results when  *
--*  requested by the HOST.                                               *
--*************************************************************************

PROC DRIVER1(CHAN OF MESSAGE dr.ot, dr.in)
  CHAN OF MESSAGE int.com1:

  INT delay, t0, pr, len:
  REAL64 e, e2, e3:
  BOOL drvr.going:
  SEQ
    delay := DRV1.DLY
    drvr.going := TRUE
    WHILE drvr.going
      SEQ
        PAR

          -- Send next packet when interpacket delay timer expires
          TIMER clock:
          INT cnt, cnt2, timer:
          [MLEN]BYTE msg:
          SEQ
            cnt, cnt2 := 0, 0
            int.com1 ? CASE                 -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE  ((DRV1.SNO > cnt) AND (delay >= 0))
              SEQ
                IF
                  DRV1.DST = RNDM           -- for randomized data test runs
                    SEQ
                      WHILE (dest[cnt2] = DRV1)
                        IF
                          cnt2 = (DRV1.SNO - 1)
                            cnt2 := 0
                          TRUE
                            cnt2 := cnt2 + 1
                      timer := timer PLUS (i[cnt2] / 1000)
                      clock ? AFTER timer
                                          -- send data packet to MUX
                      dr.ot ! ext.dat;dest[cnt2];n[cnt2]::msg
                      IF
                        cnt2 = (DRV1.SNO - 1)
                          cnt2 := 0
                        TRUE
                          cnt2 := cnt2 + 1
                  TRUE                    -- for constant data test runs
                    SEQ
                      timer := timer PLUS delay
                      clock ? AFTER timer
                                          -- send data packet to MUX
                      dr.ot ! ext.dat;DRV1.DST;MLEN::msg
                  cnt := cnt + 1
            IF                            -- notify host - sender is done!
              ((DRV1.DST <> RNDM) AND
               (DRV1.SNO > 0) AND (delay > 0))
                dr.ot ! stage.completion
              TRUE
                dr.ot ! test.completion


          -- Receive a packet and save timing data
          TIMER clock:
          INT dest, time.last, time.present:
          [MLEN]BYTE msg:
          BOOL going:
          SEQ
            t0, pr := 0, 0
            e, e2, e3 := 0.0(REAL64), 0.0(REAL64), 0.0(REAL64)
            going := TRUE
            dr.in ? CASE
              synchronize
                SEQ
                  int.com1 ! synchronize -- Resynchronize data generator
                  clock ? time.last      -- Read starting clock
                  WHILE going
                    SEQ
                      dr.in ? CASE       -- Receive driver input
                        ext.dat;dest;len::msg
                          SEQ            -- Compute/save timing values
                            clock ? time.present
                            IF
                              t0 = 0
                                t0 := time.present - time.last
                              TRUE
                                SEQ
                                  e := e + (REAL64 TRUNC(time.present - time.last))
                                  e2 := e2 + (e * e)
                                  e3 := e3 + (e * e2)
                              pr := pr + 1
                              time.last := time.present
                        report
                          going := FALSE
              terminate
                drvr.going := FALSE

          -- Report driver throughput results
          IF
            drvr.going
              SEQ
                dr.ot ! results;DRV1;DRV1.SNO;DRV1.DST;delay;t0;e;e2;e3;pr

                -- Decrease data delay (if greater than 0)
                IF
                  ((DRV1.SNO > 0) AND (delay > 0))
                    delay := delay - 10
                  TRUE
                    SKIP
            TRUE
              SKIP
:
```

```
#INCLUDE "protocol.inc"     -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"   -- test configuration parameters
#INCLUDE "DRV2_DST.inc"     -- driver packet destination array
#INCLUDE "DRV2_DLY.inc"     -- driver interpacket arrival delay times array
#INCLUDE "DRV2_LEN.inc"     -- driver data packet message length array

--*************************************************************************
--*  This process drives the DRIVER communications link.  It sends data   *
--*  packets at predetermined time intervals (set in test_setup.inc),     *
--*  receives data from the router network, and sends stage results when  *
--*  requested by the HOST.                                               *
--*************************************************************************

PROC DRIVER2(CHAN OF MESSAGE dr.ot, dr.in)
  CHAN OF MESSAGE int.com1:

  INT delay, t0, pr, len:
  REAL64 e, e2, e3:
  BOOL drvr.going:
  SEQ
    delay := DRV2.DLY
    drvr.going := TRUE
    WHILE drvr.going
      SEQ
        PAR

          -- Send next packet when interpacket delay timer expires
          TIMER clock:
          INT cnt, cnt2, timer:
          [MLEN]BYTE msg:
          SEQ
            cnt, cnt2 := 0, 0
            int.com1 ? CASE                 -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE  ((DRV2.SNO > cnt) AND (delay >= 0))
              SEQ
                IF
                  DRV2.DST = RNDM           -- for randomized data test runs
                    SEQ
                      WHILE (dest[cnt2] = DRV2)
                        IF
                          cnt2 = (DRV2.SNO - 1)
                            cnt2 := 0
                          TRUE
                            cnt2 := cnt2 + 1
                      timer := timer PLUS (i[cnt2] / 1000)
                      clock ? AFTER timer
                                            -- send data packet to MUX
                      dr.ot ! ext.dat;dest[cnt2];n[cnt2]::msg
                      IF
                        cnt2 = (DRV2.SNO - 1)
                          cnt2 := 0
                        TRUE
                          cnt2 := cnt2 + 1
                  TRUE                      -- for constant data test runs
                    SEQ
                      timer := timer PLUS delay
                      clock ? AFTER timer
                                            -- send data packet to MUX
                      dr.ot ! ext.dat;DRV2.DST;MLEN::msg
                  cnt := cnt + 1
            IF                              -- notify host - sender is done!
              ((DRV2.DST <> RNDM) AND
               (DRV2.SNO > 0) AND (delay > 0))
                dr.ot ! stage.completion
              TRUE
                dr.ot ! test.completion


          -- Receive a packet and save timing data
          TIMER clock:
          INT dest, time.last, time.present:
          [MLEN]BYTE msg:
          BOOL going:
          SEQ
            t0, pr := 0, 0
            e, e2, e3 := 0.0(REAL64), 0.0(REAL64), 0.0(REAL64)
            going := TRUE
            dr.in ? CASE
              synchronize
                SEQ
                  int.com1 ! synchronize -- Resynchronize data generator
                  clock ? time.last      -- Read starting clock
                  WHILE going
                    SEQ
                      dr.in ? CASE       -- Receive driver input
                        ext.dat;dest;len::msg
                          SEQ            -- Compute/save timing values
                            clock ? time.present
                            IF
                              t0 = 0
                                t0 := time.present - time.last
                              TRUE
                                SEQ
                                  e := e + (REAL64 TRUNC(time.present - time.last))
                                  e2 := e2 + (e * e)
                                  e3 := e3 + (e * e2)
                            pr := pr + 1
                            time.last := time.present
                        report
                          going := FALSE
              terminate
                drvr.going := FALSE

          -- Report driver throughput results
          IF
            drvr.going
              SEQ
                dr.ot ! results;DRV2;DRV2.SNO;DRV2.DST;delay;t0;e;e2;e3;pr

                -- Decrease data delay (if greater than 0)
                IF
                  ((DRV2.SNO > 0) AND (delay > 0))
                    delay := delay - 10
                  TRUE
                    SKIP
            TRUE
              SKIP
:
```

```
#INCLUDE "protocol.inc"     -- descriminated protocol for message channels
#INCLUDE "test_setup.inc"   -- test configuration parameters
#INCLUDE "DRV3_DST.inc"     -- driver packet destination array
#INCLUDE "DRV3_DLY.inc"     -- driver interpacket arrival delay times array
#INCLUDE "DRV3_LEN.inc"     -- driver data packet message length array

--*************************************************************************
--*  This process drives the DRIVER communications link.  It sends data   *
--*  packets at predetermined time intervals (set in test_setup.inc),     *
--*  receives data from the router network, and sends stage results when  *
--*  requested by the HOST.                                               *
--*************************************************************************

PROC DRIVER3(CHAN OF MESSAGE dr.ot, dr.in)
  CHAN OF MESSAGE int.com1:

  INT delay, t0, pr, len:
  REAL64 e, e2, e3:
  BOOL drvr.going:
  SEQ
    delay := DRV3.DLY
    drvr.going := TRUE
    WHILE drvr.going
      SEQ
        PAR

          -- Send next packet when interpacket delay timer expires
          TIMER clock:
          INT cnt, cnt2, timer:
          [MLEN]BYTE msg:
          SEQ
            cnt, cnt2 := 0, 0
            int.com1 ? CASE                 -- synchronize for next test stage
              synchronize
                SKIP
            clock ? timer
            WHILE  ((DRV3.SNO > cnt) AND (delay >= 0))
              SEQ
                IF
                  DRV3.DST = RNDM           -- for randomized data test runs
                    SEQ
                      WHILE (dest[cnt2] = DRV3)
                        IF
                          cnt2 = (DRV3.SNO - 1)
                            cnt2 := 0
                          TRUE
                            cnt2 := cnt2 + 1
                      timer := timer PLUS (i[cnt2] / 1000)
                      clock ? AFTER timer
                                            -- send data packet to MUX
                      dr.ot ! ext.dat;dest[cnt2];n[cnt2]::msg
                      IF
                        cnt2 = (DRV3.SNO - 1)
                          cnt2 := 0
                        TRUE
                          cnt2 := cnt2 + 1
                  TRUE                      -- for constant data test runs
                    SEQ
                      timer := timer PLUS delay
                      clock ? AFTER timer
                                            -- send data packet to MUX
                      dr.ot ! ext.dat;DRV3.DST;MLEN::msg
                cnt := cnt + 1
            IF                              -- notify host - sender is done!
              ((DRV3.DST <> RNDM) AND
               (DRV3.SNO > 0) AND (delay > 0))
                dr.ot ! stage.completion
              TRUE
                dr.ot ! test.completion


          -- Receive a packet and save timing data
          TIMER clock:
          INT dest, time.last, time.present:
          [MLEN]BYTE msg:
          BOOL going:
          SEQ
            t0, pr := 0, 0
            e, e2, e3 := 0.0(REAL64), 0.0(REAL64), 0.0(REAL64)
            going := TRUE
            dr.in ? CASE
              synchronize
                SEQ
                  int.com1 ! synchronize -- Resynchronize data generator
                  clock ? time.last      -- Read starting clock
                  WHILE going
                    SEQ
                      dr.in ? CASE       -- Receive driver input
                        ext.dat;dest;len::msg
                          SEQ            -- Compute/save timing values
                            clock ? time.present
                            IF
                              t0 = 0
                                t0 := time.present - time.last
                              TRUE
                                SEQ
                                  e := e + (REAL64 TRUNC(time.present - time.last))
                                  e2 := e2 + (e * e)
                                  e3 := e3 + (e * e2)
                            pr := pr + 1
                            time.last := time.present
                        report
                          going := FALSE
                terminate
                  drvr.going := FALSE

          -- Report driver throughput results
          IF
            drvr.going
              SEQ
                dr.ot ! results;DRV3;DRV3.SNO;DRV3.DST;delay;t0;e;e2;e3;pr

                -- Decrease data delay (if greater than 0)
                IF
                  ((DRV3.SNO > 0) AND (delay > 0))
                    delay := delay - 10
                  TRUE
                    SKIP
            TRUE
              SKIP
:
```